AFRL-RI-RS-TR-2009-258
Final Technical Report
November 2009

EFFICIENT AND SECURE REPOSITORY ACCESS

Net-Scale Technologies, Inc.

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No.AQ04/00

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2009-258 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/
STEVEN FARR
Work Unit Manager

/s/
JULIE BRICHACEK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
NOVEMBER 2009	Final	June 2008 – June 2009

4. TITLE AND SUBTITLE

EFFICIENT AND SECURE REPOSITORY ACCESS

5a. CONTRACT NUMBER
FA8750-08-C-0219

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62304E

6. AUTHOR(S)

Urs A. Muller, Alan Plofker, Chris Crudele, Paula Muller, and Beat Flepp

5d. PROJECT NUMBER
C4IE

5e. TASK NUMBER
06

5f. WORK UNIT NUMBER
03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Net-Scale Technologies, Inc.
281 State Highway 79
Morganville, NJ 07751-1157

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISE
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2009-258

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 88ABW-2009-4838, Date Cleared: 16-Nov-2009

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The principal contractor, in collaboration with the Corporation for National Research Initiatives (CNRI) demonstrated interoperability and secure sharing between three Digital Object Repository (DOR) and between the DOR repositories and a legacy system as specified in the Use Case of this project. Documentation was delivered on the architecture and the protocols in sufficient detail to allow an independent third party to develop its own DOR implementation and/or to develop a DOR interface to a legacy system. Net-Scale's efforts focused on: a) design and implement an access client for the Digital Object Architecture (DOA) repository created by CNRI, b) design and implement a functional and performance test suite for the CNRI software, and c) analyze the initial test results, identify possible weak points, and make recommendations for improvements.

15. SUBJECT TERMS
document repository; shared workspace; document collaboration; shared documents; digital object archive; uniform repository interface; secure repository access

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Steven Farr
U	U	U	UU	84	19b. TELEPHONE NUMBER (*Include area code*) N/A

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. PROJECT GOAL

In the context of a collaboration project carried out with the Corporation for National Research Initiatives (CNRI), which goal was to demonstrate interoperability and secure sharing between three DOR repositories and a legacy system, Net-Scale's efforts focused on: a) design and implement an access client for the Digital Object Architecture (DOA) repository created by CNRI, b) design and implement a functional and performance test suite for the CNRI software, and c) analyze the initial test results, identify possible weak points, and make recommendations for improvements. In addition, Net-Scale assisted CNRI to design, test, document and demonstrate the functionality developed within the scope of the collaboration project.

2. BACKGROUND

The DOA repository created by CNRI uses new and innovative paradigms for digital object storage, management, persistent identification and sharing. This requires the creation of new access client software as existing clients do not support the unique advantages of the DOA.

3. PROJECT OVERVIEW

This collaboration project with CNRI was an effort to demonstrate, on June 30th 2009, that the DO Architecture can support interoperability and secure sharing across administrative domains and with legacy systems.

In summary, there are four repositories, three of which are Digital Object Repositories (DOR1, DOR2 and DOR3) and a legacy Subversion repository (SVN). There is a collaboration space where users from the different repositories can have access to files originated in any of the systems. The shared access mechanisms are of two types: a) by use of the DOR Identity Management and b) by use of Negotiated Access among repositories. Type a) is used by DOR1 and DOR2, while type b) is used by DOR3 and SVN. For SVN to communicate with the DOR repositories, a DOR interface over HTTPS is required. For users to operate the systems, they will have the choice of two access clients: a) Java client and b) zero-footprint web client.

As stated in the Statement of Work of the collaboration project, the required capabilities for the overall system were as follows:

1. Create a “shared workspace” between the four systems.
2. Allow users to deposit files/objects in the workspace or remove them, provided that they have the necessary privileges to do so.
3. Be able to remove access to a workspace without deleting it.
4. Be able to delete a workspace.

In addition to the capabilities described in the use cases, there was a parallel effort for developing a complete test suite, including stress and performance tests as well as functional tests, which will increase the reliability and robustness of the DO Repository both in this project and in its use in any future DARPA projects.

Finally, in preparation for the DOSR DARPA program, five use cases were explored at the conceptual level. The result of this work resulted in documentation of the architectural and protocol extensions to the DO Architecture in support of these use cases.

3.1. Operational Use Case

Assume four repositories. Three are Digital Object Repositories (DOR1, DOR2, and DOR3), the other, SVN (a popular open source version control system, http://subversion.tigris.org/), is not. The owners of the repositories want some of their users to be able to collaborate by having shared access to files originating on any of the systems. The repository owners want to give the users who originate files the ability to allow or disallow different types of access, e.g., read/write, to those files.

The owners of SVN are willing to add a service to their repository that can communicate over DOR protocols or DOR APIs. Basically, they want their system to be a client of the DOR network and proxy access for their users. Their preference is for the network application layer protocol to be HTTP, as this removes problems dealing with their organization’s firewall policies.

Two of the DOR repositories (DOR1 and DOR2) are in favor of using DOR Identity Management and object sharing, and agree to do so among themselves. The DOR3 owners and the SVN repository owners are reluctant to adopt the DOR's Identity Management and object to sharing schemes. They do not want their users to have accounts with other repositories. Furthermore, they do not want other repository users to access their repository directly but they are willing to let other repositories access their repository with prior negotiation. They suggest that each system authenticate its own users and validate their access privileges, and that the systems cooperatively maintain a virtual view of the files/objects so that it appears to the users that the files/objects reside on their home system. This setup is illustrated in Figure 1.

The required capabilities for the overall system are as follows:

- Create a "shared workspace" between the four systems.
- Allow users to deposit files/objects in the workspace or remove them, provided that they have the necessary privileges to do so.
- Be able to remove access to a workspace without deleting it.
- Be able to delete a workspace.

The underlying operations of the collaboration should be transparent to the users. The repositories cooperate over the DOR protocols to maintain consistency between the data on the systems.

- If a digital object is added to the workspace, it appears instantly on all four systems.
- If a user deposits a new version of a digital object in the workspace, all four systems should immediately "see" the new version.
- The digital object appears to be a regular file on the SVN, and a regular digital object on the DOR repositories.
- If a digital object is removed from the workspace by its owner, it disappears for all other users, but not for the owner.
- If a file is deleted while in the workspace, it disappears completely from all systems.

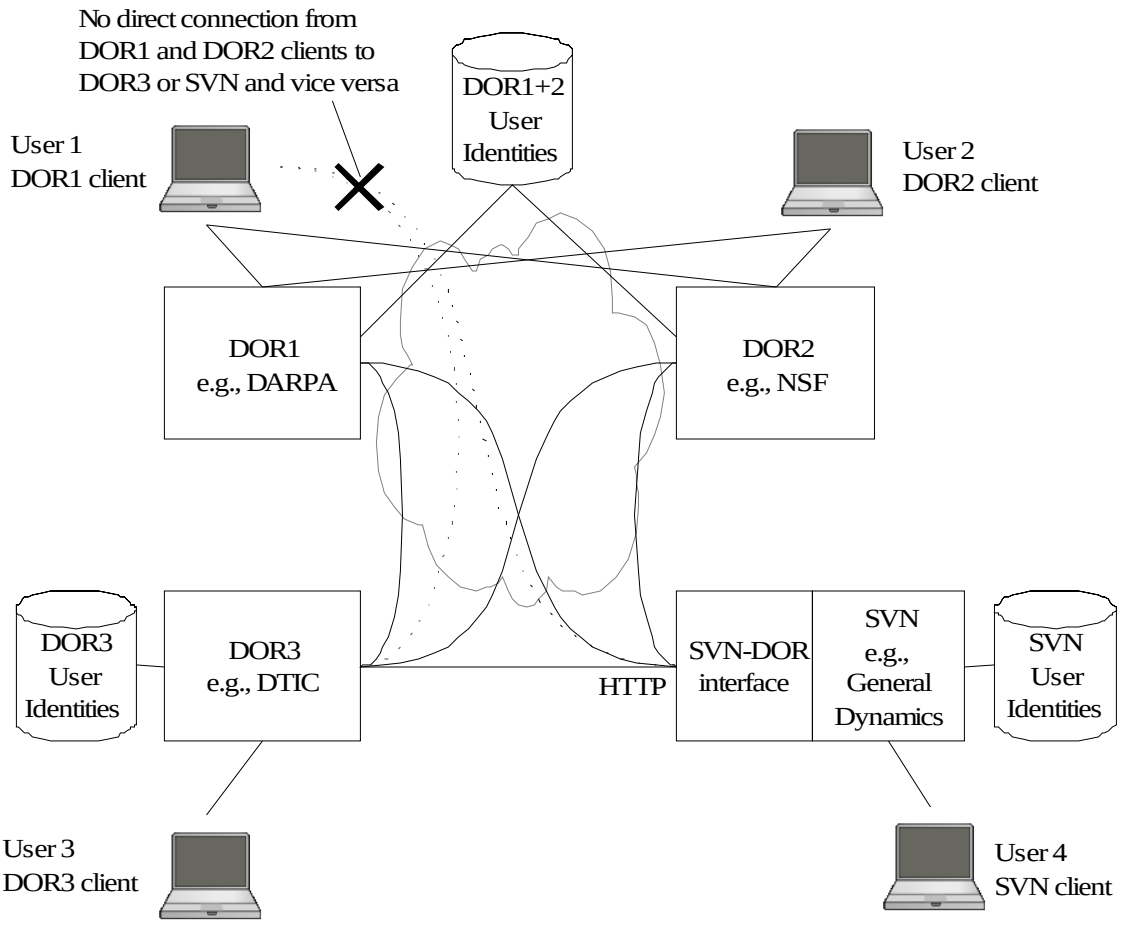


Figure 1: Interoperability and secure sharing between four repositories.

4. HIGH-LEVEL PROJECT PLAN

4.1. Interoperability and Secure Sharing Demo Work Items

The work required to extend the current DOR system for the final demonstration in June 2009 was subdivided into multiple items that are outlined in the table below:

Table 1: Responsibilities

Work Items	Responsible
1. DOR extensions	CNRI
2. Java client extensions	CNRI
3. SVN interface	CNRI
4. Zero-footprint client	Net-Scale
5. Test suite development	Net-Scale
6. System integration	CNRI + Net-Scale
7. Functional and stress test	Net-Scale
8. Demo dry runs	CNRI + Net-Scale

Figure 2 shows the graphical time line of the above project components.

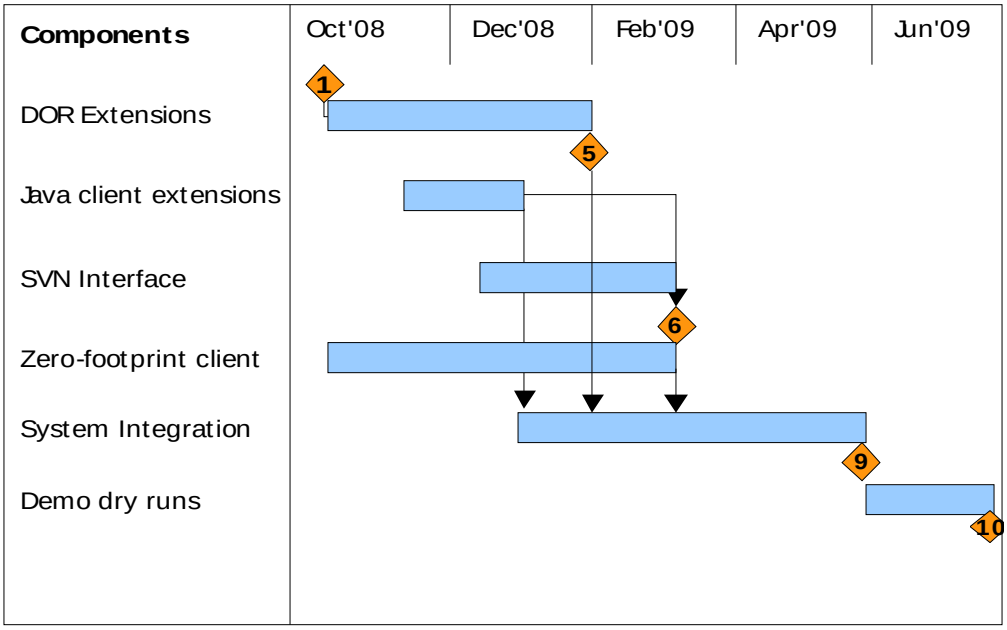


Figure 2: Graphical time line of the project components.

4.2. Testing Effort

The system testing effort involves the development of a test suite and its execution and reporting at several stages of the development. Net-Scale Technologies 's responsibilities are:

1. Development, updating, and continued application of a complete test suite for all functions of the DO Repository.
2. DO Repository documentation at both the system administration and developer levels.
3. Evolution and maintenance of the Net-Scale zero-footprint web client in the interoperability and secure sharing demonstration.

4.3. DOSR Use Cases

Lastly, we will examine and document the application of the DO Architecture, including any needed extensions, to a selection of use cases relevant to the upcoming DOSR program. This will not include any implementation work but will be done at a sufficient level of detail to guide future implementations. The specific use cases that will be explored are:

1. Resilience to network attacks (the objects are secure).
2. Provenance: tracking and being able to know who has access to information.
3. Secure search.
4. Secure sharing; the party we are sharing with does not have a DOR identity ("ad-hoc sharing", e.g., using CAC).
5. Automatic assembly of information for "commanders brief" (reuse/extend results from DARPA PAL program).

5. DELIVERABLES

5.1. Documentation of Prototype System

- Project Plan.
- High-level design document.
- Detailed design document.
- Protocol documentation (suitable for third party to use without additional information).
- Document about “lessons learned” and recommendations for future work.

5.2. Documentation of Testing Effort

- DO Repository functional analysis.
- Test suite design document.
- Functional test suite.
- Functional web client.
- DO Repository system and user administration documentation.
- DO Repository client application developer documentation.

5.3. Conceptual Analysis of the Five DOSR Use Cases

- Conceptual description of the DO Repository architectural and protocol extensions to support five additional DOSR use cases.

6. WEB CLIENT DESIGN

6.1. Background

DARPA-IPTO has requested CNRI and Net-Scale to concentrate their efforts to build a system composed of four repositories that can demonstrate interoperability and secure sharing among DOR repositories and a SVN legacy system. This chapter concentrates on the zero footprint (ZFP) web client capabilities to access the DOR repositories.

6.2. Web Client Purpose

The web client uses web technology and can be launched either through the web or locally. It does run within a web browser environment on the user's PC or laptop, and is designed to make the DOR functionality easy to use and intuitive for end users. Furthermore, the software architecture intends to make porting of the web client to hand held and wireless devices easy.

6.3. Web Client Design Overview

Figure 3 shows the ZFP web client in the context of the June 2009 demo system. This web client provides the same DO Architecture functionality as the Java client to access the DO Repository with which is connected.

Figure 4 shows the web client design. A user starts by accessing either a predefined URL or by opening a locally installed HTML file. In either case, this loads the Net-Scale web client into the browser and starts executing its code. The web client consists of static code including Cascading Style Sheets (CSS) and an initially empty HTML page. All functionality is provided by the dynamic code which includes JavaScript and a Java applet. The JavaScript code creates HTML elements dynamically and places or moves them in the browser screen to create the desired user interface and effects. The CSS code defines the graphical attributes of the HTML objects. The JavaScript code further interacts with the Java applet.

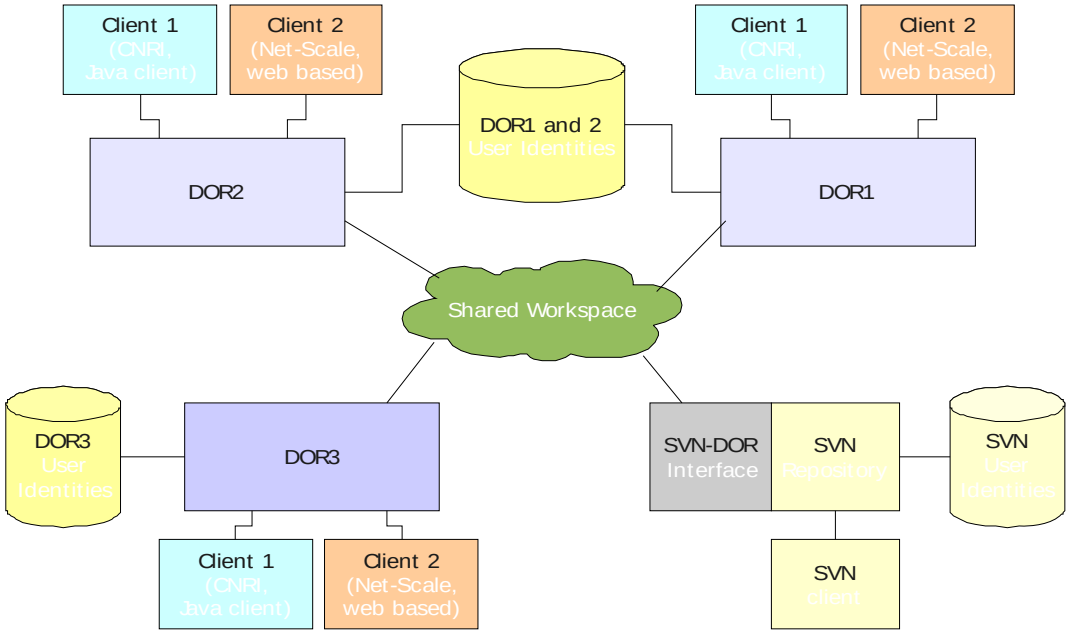


Figure 3: Clients accessing DO Repositories.

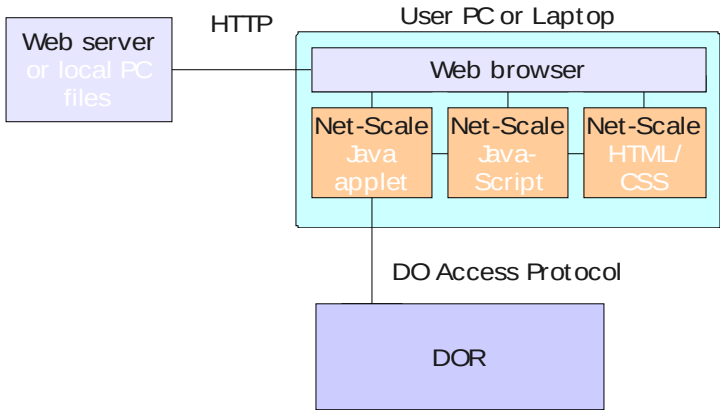


Figure 4: Net-Scale client design using web technology.

Note, that no traditional web server applications, such as CGI scripts, are used, since all necessary software is embedded in the start web page and related files. This allows a user to start the client from the local file system without requiring a web server to be available.

6.4. ZFP Web Client Functionality Overview

The main web client functionality will be:

- Deposit a Digital Object.
- Retrieve a Digital Object.
- Remove a Digital Object.
- Deposit a Digital Object in shared workspace.
- Retrieve a Digital Object from shared workspace.
- Remove a Digital Object from shared workspace.
- Search, which includes the home repository and the shared workspace.
- Access the local file system.

7. REQUIREMENTS

- Demonstrate that the ZFP web client can access full functionality of the Digital Object Repository supporting interoperability and secure sharing.
- Test functionality of ZFP client.

8. JAVA APPLET

8.1. Purpose

The Java Applet provides processing capabilities in the local device or PC and the interface to the local resources as well as to the DO Architecture layer through the DO Protocol and DO Repository APIs. Using Java technology permits portability across platforms. Moreover, encapsulating the processing in an applet avoid the requirements of client software installation.

8.2. Design

Figure 5 shows the Java Applet design. This Java Applet has three main components: a) japnsrep, b) jarnslfs, and japnsrep. Japnsrep interact with the Javascript to capture and signal user interactions and display of information. Jarnslsf provides all the processing and access to the local file system, and Jarnsrep provides all the interfaces to the DOR API, and therefore interaction with the DO Architecture layer.

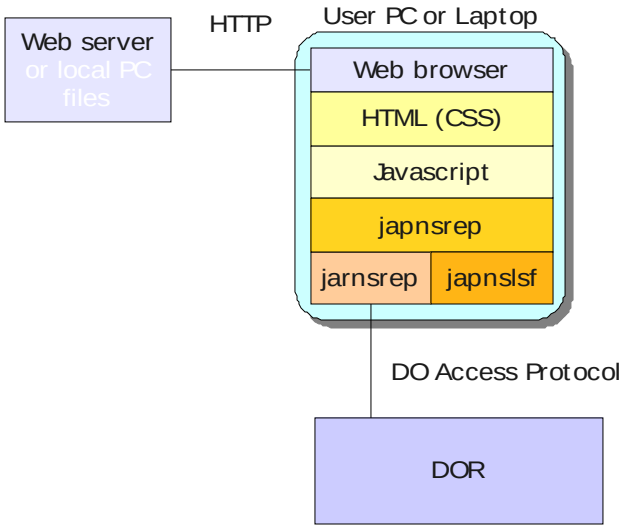


Figure 5: Net-Scale Java Applet design.

9. FUNCTIONAL LIST

This is the group of basic functions. They are based on the Use Cases in the *Statement of Work – DOR Interoperability and Secure Sharing Demonstration.* The testing will be performed in two stages: first on the existing functionality, and second on the new functionality developed for the DOR09 use cases.

9.1. Stage 1

The following table indicates existing functionality.

Table 2: Functional Test Groups

Function	***Group***	***Operand***			
		PDO	***DO***	***DE***	***DA***
1.	*Connect to repository*	**X**			
2.	*Create*		**X**	**X**	**X**
3.	*Modify*			**X**	**X**
4.	*Access (Search)*				**X**
5.	*Retrieve*			**X**	**X**
6.	*Delete*		**X**	**X**	**X**

9.2. Stage 2

In addition to the new functionality the following will be performed:

- Regression: Perform the Stage 1 tests on a DOR SVN repository.

10. TEST METHODOLOGY

10.1. Expected Results

There are two distinct types of test: those expected to succeed and those expected to fail. The latter are used to test the error conditions.

10.2. Data Set

The file types will include (but are not limited to): avi, mov, mp4, pdf, odt, doc, txt, and jpg. There will be files of varying size from 1 byte to 50Mb within each file type. Larger file sizes (up to 4.5 GB) will be tested as part of the functional test plan. Each file will have the correct mime type.

11. TEST ENVIRONMENT

This section describes the network and system configuration, along with the tools used to test the system.

11.1. Network Configuration

These tests will be run from the Net-Scale location, over the Internet, to the dnanetscale server in the CNRI network.

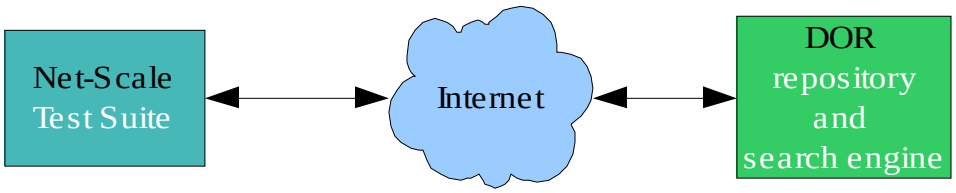


Figure 6: DOR - Net-Scale Test Environment

11.2. Test Tools and Interfaces

The Net-Scale Test Suite is written in Perl. It interfaces with the DO Repository as shown in the following diagram.

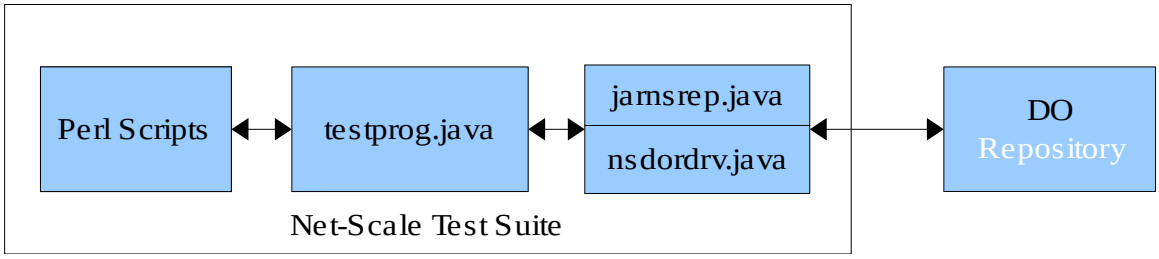


Figure 7: Net-Scale Test Suite Interface to DO Repository

12. USE CASES

These use cases identify a test group. For each use case there will be one or more test cases.

12.1. Functional Use Cases

Table 3: Functional Use Cases

Use Case ID	Use Case Description	Notes	Expected Results
1.	Authenticate	A listOperations call is made which establishes a connection to the server. It is then disconnected. Authenticate is implicitly tested in all other cases.	User is connected then disconnected.
2.	Deposit a file	Deposit a new randomly selected file, 'f1'. This test performs many operations, including create DO, with default DA, and writes to DE elementid=content. Use files of different mimetypes and sizes.	DO is created with DE with elementid=content f1, and default DAs.
3.	Add attributes to the DO created in #2	Set attributes: notes, publisher, creator, etc. Creates new attribute if needed.	Attributes are updated and can be retrieved via Search
4.	Modify the local file	Replace the content of the data element in an existing DO in the repository.	The elementid=content DE is replaced in the existing DO.
5.	Encrypt the deposited file	Creates a new DO with default DA and writes to encrypted DE.	The DE is not stored in the clear.
6.	Get the encrypted file	Download the un-encrypted DE to a local file.	The DE stored in the clear

7.	Rights (permissions)- Assign	Updates a DO'sDE internal.rights to allow user's read operation.	The other user can access and read f1.
8.	Rights (permissions) - Remove	Updates a DO'sDE internal.rights to remove user's read operation.	The other user can no longer access or read f1.
9.	Search for specific attribute value	Specify search query for particular attribute and value. For example, filename.	Reference to DO that matches the query. The reference includes the Object ID and the attributes.
10.	Search for keywords	Search for keywords in file content (mimetypes supported by Lucene). The index is created from output from a mimetype parser.	Reference to DO that matches the query. The reference includes the Object ID and the attributes.
11.	Download a file	Download a file. Use files of different mimetypes and sizes.	File is stored locally and can be read.
12.	Remove attributes	Remove all attributes from data element created in test case #3	Search for DO by removed attribute returns no results.
13.	Remove file	Remove object with data element and attributes	DO cannot be found.

13. TEST CASES

13.1. Functional Test Cases

Table 4: Functional Test Cases

Use Case ID	Use Case Description	Test Case ID	Expected Results
1.	Authenticate	authenticate	Credentials are authenticated and connection is established and disconnected.
2.	Deposit a file	put	DO is created and writes to DE content f1, and default DAs.
3.	Add attributes to the DO created in #2	addDOAttribute	Sets the value of an attribute (creating a new attribute if needed).
4.	Modify the local file	mod	The DE content is replaced in the existing DO.
5.	Encrypt the deposited file	putEncrypted	Creates new DO with default DA and writes to encrypted DE.
6.	Get the encrypted file	getEncrypted	Downloads unencrypted DE to a local file.
7.	Rights (permissions)-Assign	allRead	Internal.rigths are modfied. The other user can access and read the DO data element
8.	Rights (permissions) - Remove	revokeRead	Internal.rigths are modfied. The other user can no longer access or read the DO data element.
9.	Search for specific attribute value	Specify search query for particular attribute and value. For example, filename.	Reference to DO that matches the query. The reference includes the Object ID and the attributes.

10.	Search for keywords	Search for keywords in file content (mimetypes supported by Lucene).	Reference to DO that matches the query. The reference includes the Object ID and the attributes.
11.	Download a file	get	File is stored locally and can be read.
12.	Remove attributes	deleteDOAttribute	Sets the value of an attribute to null. Search for DO by removed attribute returns no results.
13.	Remove file	delete	DO is removed and cannot be found.

14. INITIAL TEST EXECUTION

14.1. Demonstration of Test Suite Functionality

The initial tests have been executed for this milestone. The tests were run on the Net-Scale LAN, with a local installation of the DO repository. The test execution is for a single user. The test configuration specifies the test duration (hours) and test interval, or epoch. Data is collected for each interval. For the first test the duration was 10 hours with 1 hour intervals. For the second test the duration was 12 hours and the interval was 2 hours.

This version of the test suite includes get, put, and delete operations on very small files. Note that each operation performs a connect (authentication) and disconnect. The test suite will be modified to allow multiple operations (get, put, delete, etc) over a single connection, thereby removing the connection overhead.

These test results are preliminary; they are presented here to show operation of the test suite. No conclusions are to be drawn with regards to the performance or operation of the DO repository.

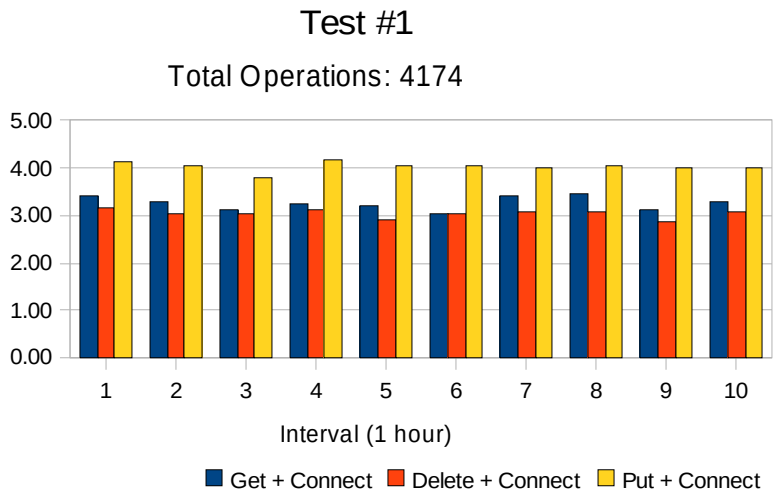


Figure 9: Test #1

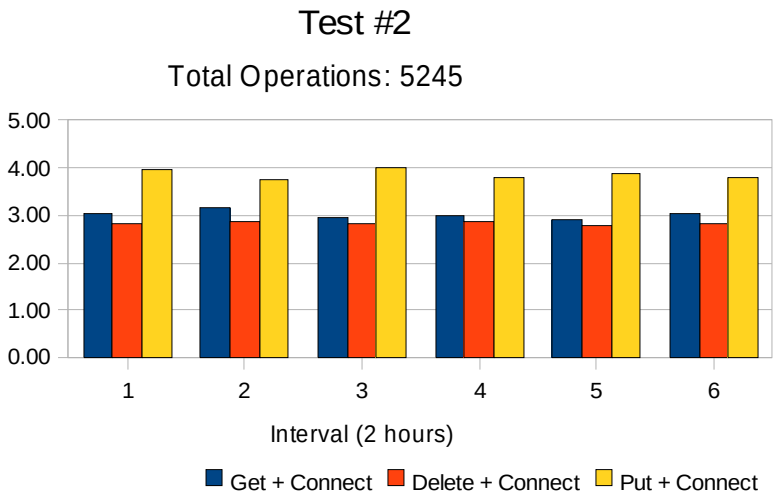


Figure 8: Test #2

This test run included test cases for connect (includes authentication) and modify file (data element content). The times reported for each operation include the connect and disconnect times. That is, get + connect, delete + connect, put + connect, modify + connect.

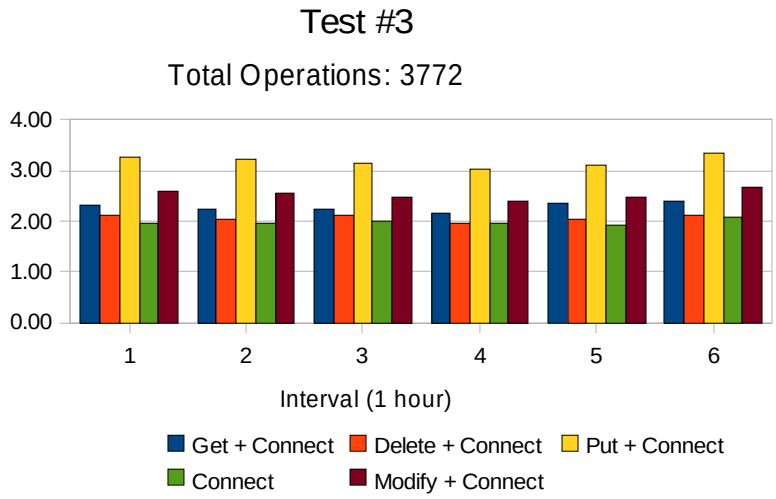


Figure 9: Test #3

The following chart shows the minimum and maximum times per operation, including connect times.

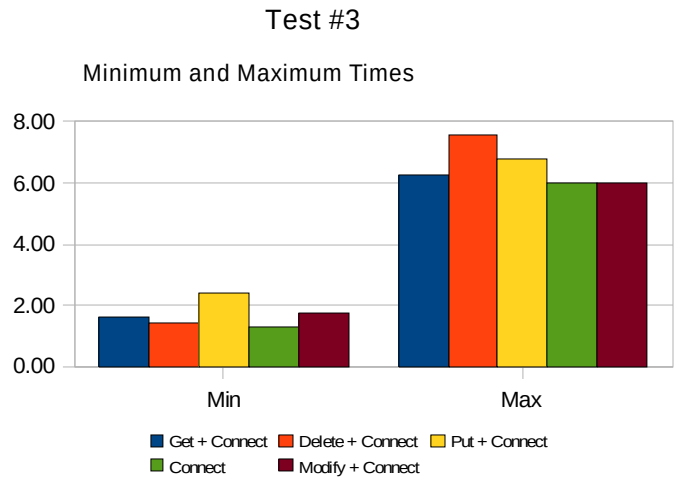


Figure 10: Minimum and Maximum Times Per Operation.

15. TEST SUITE METHODOLOGY

15.1. Overview

The primary objectives of the test suite are to validate the functional integrity of the DOR operations and collect operation timing. The test scripts were designed with the following concepts.

- Operation list: this list specifies the operations (or test cases) to be performed. In the standard test scenario, entries are chosen randomly from this list. Given a balanced representation and a sufficiently long test run, there is approximately an equal distribution of operation executions. This distribution can be modified by changing the number of times an operation is represented in the list. For example, this list specifies test cases for authenticate, put, get, and delete:

  ```
  @opl = ($conf->{name_auth},
  $conf->{name_put},
  $conf->{name_get},
  $conf->{name_del})
  ```

  To double the files added to the repository simply repeat the entry for that test case:

  ```
  @opl = ($conf->{name_auth},
  $conf->{name_put},
  $conf->{name_put},
  $conf->{name_get},
  $conf->{name_del})
  ```

  When an operation is selected the arguments to the operation are validated. If the validation fails the operation is skipped.

- Data validation: local shadowing. When a file is deposited an object is created and data elements and attributes are assigned. The same information is maintained in the local file system (a shadow copy). Subsequent changes to the object are made to the local shadow copy. At the conclusion of the test all objects in the repository are compared to their local shadow to verify data integrity.

- Simultaneous users: A parameter in the configuration file specifies the number of users. Asynchronous threads are created and executed to represent simultaneous users. Each thread uses the same configuration information but a unique private key to simulate independent users.

- Replay (repeat test execution): Post processing of the log file is used to create a replay transcript. This transcript identifies the operations and data used by the test program. The transcript is then used as input to test-dor09.pl to carry out the exact sequence of operations with the same source data as the original test run.

15.2. Configuration

The following parameters are defined in testconfig.pm. This is not the exhaustive list, but those most likely to be modified. Refer to the comments in the module for more information.

Table 5: Parameters in Testconfig.pm

Parameter	Description
server	The handle for the repository and indexer. Ex: "cnri.test.sean/netscale1:cnri.test.sean/netscale1idx"
copylist	The list of external files to be used by those operations that require source files.
opl	The operation list. Comma-separated list of strings corresponding to the test cases.
users	Number of simultaneous users.
epochs	Number of test intervals.
duration	Length of each epoch, in seconds.

15.3. Operations

The following operations are tested. They are specified in the operations list. Each user thread authenticates with the repository once; test case execution continues under this single authentication until the test ends. The authenticate test case is an exception.

1. Authenticate and connect to the repository.
2. Write (deposit) a randomly selected file.
3. Set an object attribute.
4. Modify a randomly selected file.

5. Write an encrypted file
6. Read an encrypted file
7. Set permissions – allow (rights).
8. Set permissions – revoke.
9. Read (retrieve) a randomly selected file, i.e, the content data element.
10. Delete an attribute.
11. Delete a randomly selected object.

15.4. Test Results

During a test, the execution time of each operation is recorded to a log file. Post processing produces statistics such as iteration count, average time, standard deviation, etc. on a per epoch per operation basis as well as collectively for the test run. Multiple log files may be processed to produce collective statistics over multiple test runs.

16. TEST SUITE ARCHITECTURE

16.1. Overview

The test suite interfaces with the DO Repository through Java libraries which contain the DO API. The components are shown in Figure 11.

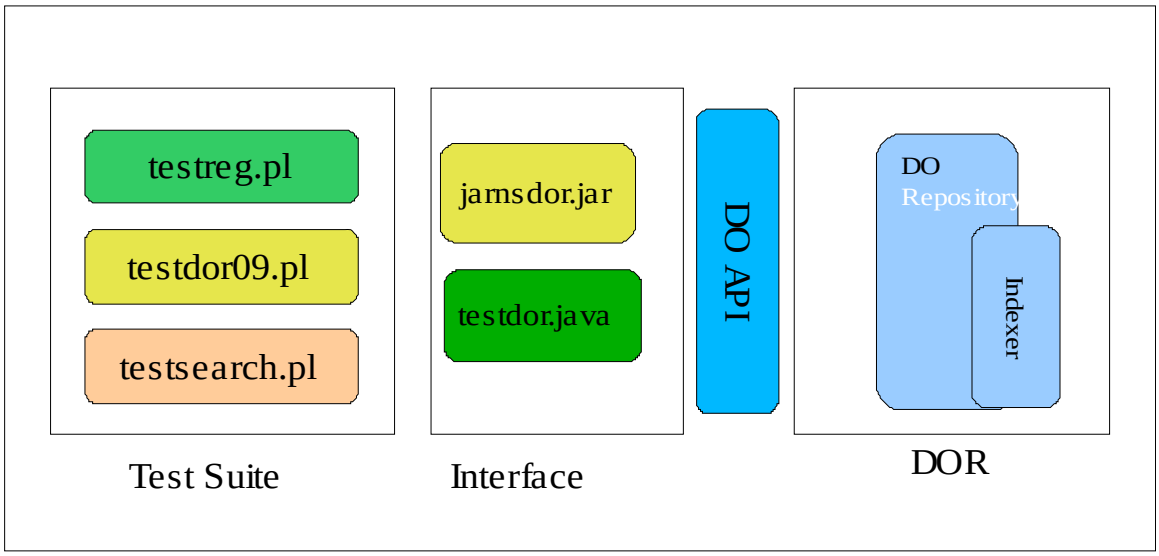


Figure 11: Test Suite Architecture

These components are included in the source and test script distribution. The directory structure and files are referenced throughout this document.

16.2. Modules

The test suite is written in Perl and interfaces to the DO API via Java libraries. The test suite is comprised of the modules shown in Figure 12.

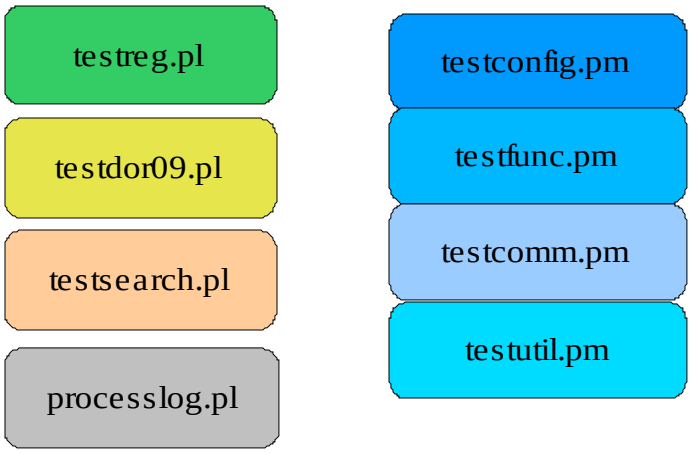


Figure 12: Test Suite Modules

These modules are used as follows.

Table 6: Test Suite Modules

Module	Description
testutil.pm	Low level functions used in other modules. These functions have no dependencies beyond standard Perl modules.
testfunc.pm	This module contains the test case functions called from the Perl scripts. It also contains the functions that interface with the java testprog.
testcomm.pm	This module contains the functions which provide interprocess communication. Each user thread sends information to the Java test program, including the next command to execute, and receives status and result information.
testconfig.pm	This module contains the configuration data for the test suite. It includes paths, initial variable values, and the operations to be tested.
testreg.pl	This is the main test script for the individual functional test cases, corner (exception) cases, and error cases. It is used for regression testing.
testdor09.pl	This is the main test script for performance and data integrity tests.
testsearch.pl	This is the search test script. It uses the same control mechanism as testdor09.pl but only invokes search test cases.
processlog.pl	This script post processes the log files from one or more test runs and creates an html page with the test results.

16.3. Test Data

The test suite operates with external test files and user data. The test files can be generated automatically, and/or specific files can be used. This allows for files of various MIME types and sizes to be used.

The files are located in the following directory structure:

src/

certificates/ : the user private key files. The files are named u<number>_priv_key, where <number> is 01..max number of users. The password is “user<number>”.

testfiles/ : files used by the test script for deposit.

16.4. Test scripts

These test scripts are described in the table above, and are found in the source tree:

src/

testscripts/ :
processlog.pl, testdor09.pl, testreg.pl, testsearch.pl
testcomm.pm, testfunc.pm, testutil.pm, testconfig.pm

16.5. Java Interface Modules

The test cases in testfunc.pm invoke DO API operations via the testcomm.pm, testdor.java and jarnsdor.java modules. The testcomm.pm module executes testdor.java in a separate process and provides a mechanism for communicating with the active process. The testdor.java module caches an instance of the access library jarnsdor.java for reuse and streamlines the DO API for testing purposes (meaning, testdor.java takes care of file streams and buffered reading, etc...). The library jarnsdor.java directly makes DO API calls while maintaining repository connection state.

These Java modules are located in the source tree:

src/

jarnsdor/

16.6. Logging and Log Files

16.6.1. Logging

The test suite uses the Perl module log4perl, a Perl port of log4j. This module allows messages to be assigned to logging levels DEBUG, INFO, WARNING, ERROR, and FATAL. The log messages may be written to stdout or a file. Filenames are created based upon the *script_name_date_timestamp*. Log files are stored in the directory src/test-results/*timestamp*/. This allows many tests to be run without overwriting existing logs.

16.6.2. Log Files

These guidelines will help interpret the log file contents. They are organized by log level.

- DEBUG: These messages are for information and debugging purposes, in the event of an error. All functions report error and status. Some log input as well. All calls to the java test program are logged as well. Here are a couple of examples.
    - 2009/01/26 08:56:11 *DEBUG* testfunc::write_to:355> user1: Executing: write_rnd_file

      The function write_to() in module testfunc.pm is called from the main test script testdor09.pl.
    - 2009/01/26 08:56:21 *DEBUG* testfunc::writefile_dor:819> cmd: put cnri.test.alan/ns_repo2:cnri.test.alan/ns_repo2 cnri.test.user01/user01 /alan/svnws/prj/darpa/dor09-1.0/src/testdata/certificates/u01_priv_key user01 /alan/svnws/prj/darpa/dor09-1.0/src/testtemp/testfiles/1.dat

      The function writefile_dor called the java testprog (indicated by cmd:) for the put operation with the shown credentials and file to deposit.
- INFO: These informational messages describe the current operation.
    - 2009/01/26 08:56:11 *INFO* testfunc::write_to:346> Done executing test data file creation script: /alan/svnws/prj/darpa/dor09-1.0/src/testscripts/createTestdata.pl
    - 2009/01/26 08:56:11 *INFO* testfunc::write_to:346> user1: Initializing for </alan/svnws/prj/darpa/dor09-1.0/src/testdata/certificates/u01_priv_key>

- WARN: These informational messages do not indicate an error, but rather a condition is not as expected.
    - 2009/01/26 08:56:11 *WARN* testfunc::write_to:337> user3: Not mirroring

      Local mirroring is the default configuration. This message indicates no local mirroring and verification will be performed for user3.
- ERROR: An operation or test script failed. At this point the script will terminate.
    - 2009/01/26 09:04:04 *ERROR* testfunc::check_stdout:3370> Error: class testdor$NSTestException

      2009/01/26 09:04:04 *ERROR* testfunc::list_objects:1231> a test command reported an error!

      2009/01/26 09:04:04 *ERROR* testfunc::write_rnd_file:2437> Cannot list objects - return value:1

17. HOW TO RUN THE TEST SUITE

17.1. Edit the Configuration

Determine the test cases to be executed, the number of simultaneous users, the number of epochs, and their duration.

Edit the Perl module testconfig.pm and set these parameters.

Table 7: Frequently Edited Parameters in Testconfig.pm

Parameter	Description	Default Value
$conf->{server}	The repository to test. Set to repository_handle:indexer_handle	No default.
@copylist	List of files to deposit in the repository.	
@opl	List of test cases to execute.	
$conf->{users}	Number of simultaneous users.	10
$conf->{epochs}	Number of test cycles.	12
$conf->{duration}	Duration of a test cycle, in seconds.	3600

17.2. Initialize the Environment

Verify the following before running the test script.

- Repository is running and accepts connections from the workstation running the tests (the test workstation).
- The test workstation has sufficient free disk space for the log files and local shadow directories. This will be determined by the length of the test and the number of users.

17.3. Run the Scripts

To run the script,

```
cd src/jarnsdor
make testdor
```

17.4. Interpret the Results

The log file post processor, *processlog.pl,* is used to collect the results from one or more test runs. If the script detects multiple log files it creates a total of all the operations.

The parameter $conf->{auto_process} in *testconfig.pm* is used to run the post processor automatically at the end of a test run. To run the script manually, copy the file *processlog.pl* into the same directory with one or more log files and run the script. The output file name has the same name as the log file with a *.html extension.

18. ADVANCED TOPICS

18.1. Replay

Log file replay is currently under development.

18.2. Customizing a Test Execution

This section describes the steps to configure the test suite for a specialized test. Suppose a test is to be executed that just performs deposits of large files. Perform the following steps.

18.2.1. Adding Test Files

Add the new files to the testdata subdirectory: ./src/testdata/testfiles

18.2.2. Configuring testconfig.pm

Two changes must be made:

1. Modify the *copylist* to exclude existing files and add the new files. Make a copy of the existing definition, then comment-out (prefix with #) the existing definition. Edit the new *copylist* to specify:
   my @copylist = (
   "new-large-file1",
   "new-large-file2"
   );
2. Modify the *opl* to only include:
   @opl = ($conf->{name_put})

18.3. How to Add a Test Case

This section describes the steps to add a new test case to the test suite.

There may be a one-to-one association between a test case and an operation to test. The following sections describe the test case for depositing an object. These files are all involved in a test case:

- testconfig.pm: Configuration information, including the test case ID and any required data for this case.
- testdor09.pl: Main control loop for the test case.
- testfunc.pm: This file contains the actual test case definition and execution. Most of the logging occurs here.
- testdor.java: This is the java implementation of the test case. It uses the Digital Object API to interact with the DOR. The execution is timed.

18.3.1. Configuration: testconfig.pm

At a minimum, one test case must be listed in the operation list. Update the comment section and add the case to the **opl**.

```
# set the operation distribution
# the operations are as follow:

$conf->{name_auth} = "Authenticate";
$conf->{name_put} = "Put";
$conf->{name_addDOA} = "Add DO Attribute";
$conf->{name_mod} = "Modify DE";
$conf->{name_putEnc} = "Put Encrypted";
$conf->{name_getEnc} = "Get Encrypted";
$conf->{name_allowRead} = "Allow Read";
$conf->{name_revokeRead} = "Revoke Read";
$conf->{name_search} = "Search";
$conf->{name_get} = "Get";
$conf->{name_delDOA} = "Delete DO Attribute";
$conf->{name_del} = "Delete DO";
$conf->{name_new} = "New Test Case";
```

18.3.2. Main Test Loop: testdor09.pl

Locate the switch statement in the Perl script and add case *$name_new.* The following should be added to the case:

- log file reporting, includes status and error messages.
- call the function which implements the test case.
- update counters.

18.3.3. Test Case Function: testfunc.pm

The Perl implementation of the test case is in this module. A template function has been included named *example_test_case_rnd()*. Copy this example for the new case.

The following must be performed:

- add the function name to the EXPORT list.
- create the function which builds the test case enviroment.
- create the function which executes the case (calls the java equivalent).

18.3.4. Java Test Case Function: testdor.java

This Java program is the interface to the DO API. The test case (API call) must be implemented here. Refer to the example_test_case and cmdExample() in the file.

The following must be performed:

- add the function for the test case.
- call the test case function from main().
- add the function to the command line Exception which prints Usage information.

18.3.5. Compile the Test Suite

Only the Java code must be compiled. This is done from the ./src directory by entering:

```
make clobber
make
```

19. SAMPLE TEST RESULTS

This is a sample page created from one log file.

Test Summary

Users	Iterations	Avg Time
2	161	2.913

Test Details

	Iterations	Avg Time	Std Dev	Min	Max
read_rnd_file	94	2.314	0.308	1.796	3.478
write_rnd_file	67	3.753	0.737	3.052	7.023

Per Epoch Details

		Iterations	Avg Time	Std Dev	Min	Max
read_rnd_file	**epoch 1**	11	2.174	0.354	1.796	2.982
	epoch 2	11	2.23	0.267	1.866	2.696
	epoch 3	8	2.32	0.23	1.899	2.622
	epoch 4	11	2.387	0.314	1.983	2.9
	epoch 5	16	2.296	0.2	2.005	2.686
	epoch 6	10	2.231	0.189	2.032	2.635
	epoch 7	10	2.464	0.33	2.149	3.273
	epoch 8	10	2.326	0.355	2.018	3.233
	epoch 9	4	2.23	0.292	1.971	2.644
	epoch 10	3	2.782	0.651	2.188	3.478
write_rnd_file	**epoch 1**	8	3.499	0.758	3.052	5.336
	epoch 2	7	3.93	0.829	3.22	5.106
	epoch 3	8	3.303	0.14	3.112	3.56
	epoch 4	6	3.531	0.297	3.18	3.943
	epoch 5	5	4.324	1.414	3.272	6.746
	epoch 6	6	4.419	1.354	3.392	7.023
	epoch 7	7	3.756	0.326	3.254	4.151
	epoch 8	7	3.719	0.355	3.391	4.355
	epoch 9	9	3.627	0.197	3.322	3.946
	epoch 10	4	3.809	0.63	3.437	4.75

Log files processed:
testdor09-testdor_20090126-133003.log

T

his page was created from two log files.

Test Summary

Users	Iterations	Avg Time
2	308	3.166

Test Details

	Iterations	Avg Time	Std Dev	Min	Max
read_rnd_file	175	2.606	1.659	1.796	17.81
write_rnd_file	133	3.903	1.721	2.98	21.252

Log files processed:
testdor09-testdor_20090126-135021.log
testdor09-testdor_20090126-133003.log

20. PERFORMANCE TEST PLAN

The goal at this stage was to implement a test harness and test environment and to demonstrate that it could be used to measure the performance of the repository system. This was accomplished and, in addition, preliminary test runs were used to identify and address some areas for performance improvement. Comprehensive system tuning and assessments of performance were not in the scope of this project.

21. METHODOLOGY

The goal for the test program was to simulate repository performance under continuous use by a different numbers of users, with a focus on upload, download and search operations.

The test program will upload/download/search files of different sizes for a specified duration. Virtual users will simulate concurrent operations. There are four test groups:

1. Baseline – A single file will be uploaded by 1 user, 5 users, and 50 simultaneous virtual users; each virtual user will continuously upload the file for the test duration of 60 minutes. This scenario will be repeated for download and search. Search will be performed by filename and keyword. There will be separate test runs with specific file sizes.
2. Random data – These tests use a collection of various file types of sizes ranging from 1 byte to 100MB. The test will be executed as above: upload for 1, 5, and 50 virtual users, then download, and search.
3. Random data/random operations – using the random data set above, upload/download/search will be randomly selected. This test uses 1, 5, and 50 virtual users and runs for 60 minutes.
4. Duration – Group #3 will be run with 5 virtual users for 1, 5, and 10 days.

Note that there are many operations which are not included in these performance tests – they are included in the functional/feature test cases.

21.1. Data Set

The file types will include (but are not limited to): pdf, odt, doc, txt, and jpg. There will be files of varying size from 1K bytes to 11MB within each file type. Larger file sizes (up to 4.5 GB) will be tested as part of the functional test plan. Each file will have the correct mime type.

21.2. Test Results

If a test does not run to completion the test results are considered inconclusive. Upon successful conclusion the test reports the average time per operation, per user. The files uploaded and downloaded will be logged. The following will also be monitored: CPU usage, memory usage, disk space usage.

22. TEST ENVIRONMENT

This section describes the network and system configuration, along with the tools used to test the system.

22.1. Network Configuration

The tests will be run in a LAN and WAN configuration.

22.2. Test Tools

The tests are written in Perl and Java. Standard Linux programs are used to monitor performance (top), network utilization (ntop), and disk i/o (iotop).

These tools consumed significant CPU resources so the client and server workstations were monitored periodically.

23. TEST CASES

All of the following cases will be executed on the final release of DOR09, although they will be run throughout the project.

Table 8: Test Case Matrix

Test Case Id		Operation	Users	Data	Duration (min)	Test Executed (Y/N)
Group	Case					
1 Baseline	A	Upload	1	11MB file	60	Y
	B	Upload	5	11MB file	60	Y
	C	Upload	50	11MB file	60	Y
	D	Download	1	11MB file	60	Y
	E	Download	5	11MB file	60	Y
	F	Download	50	11MB file	60	Y
	G	Search	1	11MB file	60	Y
	H	Search	5	11MB file	60	Y
	I	Search	50	11MB file	60	Y

2 Random Data	A	Upload	1	Random	60	Y
	B	Upload	5	Random	60	Y
	C	Upload	50	Random	60	Y
	D	Download	1	Random	60	Y
	E	Download	5	Random	60	Y
	F	Download	50	Random	60	Y
	G	Search	1	Random	60	Y
	H	Search	5	Random	60	Y
	I	Search	50	Random	60	Y

3 Random Data / Random Operations	A	Random	1	Random	60	Y
	B	Random	5	Random	60	Y
	C	Random	50	Random	60	Y

4 Duration	A	Random	5	Random	14400 (1 day)	Y
	B	Random	5	Random	72000 (5 days)	Y
	C	Random	5	Random	144000 (10 days)	N

24. TEST CASE RESULTS

24.1. Hardware Configuration

These workstations were used for the tests.

Table 9: Client Workstation Configuration

Client Laptop #1 Dell Precision M65	
Operating System	Ubuntu 2.6.24-23 Hardy Heron Workstation Gnome desktop i686
Processor	Intel(R) Core(TM)2 CPU T7400 @ 2.16GHz
Memory	2074212 KB
Disk	60Gb

Client Laptop #2 Dell Precision M65	
Operating System	Ubuntu 2.6.28-11 Jaunty Jackalope Workstation Gnome desktop i686
Processor	Intel(R) Core(TM)2 CPU T2400 @ 1.83GHz
Memory	1025152 KB
Disk	60Gb

Table 10: Server Configuration

Repository “no name” Server	
Operating System	Ubuntu 2.6.24-23 Hardy Heron Server 8.04 x86_64, updated to Ubuntu 2.6.28-11 Jaunty Jackalope Server 9.04 x86_64
Processor	Intel(R) Pentium(R) Dual CPU E2180 @ 2.00GHz
Memory	1025136 KB
Disk1	80Gb
Disk2	1T

24.2. Observations and Notes

All of the test cases were executed using the DOP and repository version do-1.17-7.

24.2.1. Test Execution Methodology

An attempt was made to maintain consistency in the test environment, however this was not always possible. For example, some of the upload tests were executed on an empty repository, but download and search required a populated repository. The configurations and environment were frequently monitored.

- The clients and server are connected to the same network switch. There was no other LAN traffic affecting the test systems.
- The tests were only executed in a LAN configuration. Internet access was required for repository handle resolution only.
- A repository was created on the server for the users PDOs. WAN traffic was limited to handle resolution, which occurred only during the test script initialization.
- The server disk space, CPU and memory utilization were monitored.
- The size of the repository logs were checked, as well as errors in the error log.
- The status of the indexer was determined by verifying the timestamp value in status.dct against the day's transaction log. If the timestamp was not the last entry in the log (and stayed that way), the indexer was not operational.

- The repository was restarted when an error occurred, or if performance degraded significantly. The server was rebooted infrequently.
- The test script output is a log file containing all test activity and results. This log is post-processed for the test statistics.
- The throughput was calculated for the upload and download operations. This measurement was performed at the application level and excludes protocol overhead and retransmitted data packets. This is referred to as *goodput* (see http://en.wikipedia.org/wiki/Goodput). This was computed and reported for each test user.

24.2.2. Observations

The following were observed during test execution.

- The tests were not run over SSL (the SSL port was not included in the repository handle). Standard DOP encryption was used.
- The repository thread resource pool is increased during test execution and drops when the test is completed. The thread pool increases significantly for the 50 user tests, and does not return to previous levels. For example, when the repository is initialized, the count is approximately 31; during a test it increases to approximately 70, then drops to the 50s when the test completes. At times the counts were observed to be as high as 216.
- Server CPU utilization was between 15 and 55%, depending on the test and number of users. Most of the observations were in the low 20%.
- A few times an error occurred which generated copious amount of error output that caused an out-of-space disk condition. Freeing up space generated more error output. The repstat.dct and status.dct files had 0 bytes. This prevented the repository from starting and it had to be re-initialized.
- The indexer continuously processes the transaction log. When the repository is modified (upload and delete operations) the indexer would get behind, but never completely stop with this software release. The impact of the indexer on the test results is unknown. It would be good to run the upload and download tests with the indexer disabled.
- For some of the tests, the standard deviations for the search times are greater than the actual times (2-3 fold).
- Search results are out of date while the indexer catches up. This was observed during the 1 and 5 day duration tests.
- The size of the index is an unknown variable in the search tests.
- The Lucene version integrated in the repository is out-of-date.
- There can be a large variability in test execution times (as indicated by the standard deviation) and between test executions.
- The time for handle resolution is not included in the test results. In this implementation, handle resolution only occurs during authentication and not for each object request. The recorded times are for the operation being tested.
- A number of test runs were attempted before the ones reported in this document. A number of issues were identified - some have been resolved and others are under investigation.

24.3. Test Results

These tests were run in a LAN configuration using do-1.17-7 distribution. Each test ran for 60 minutes. DOP over SSL was not used for these tests.

The first column in the test case tables labeled Group / Case refers to the test case identified in Table 13. The suffix refers to the execution of that test case. For example, 1A-1 refers to the first test run of case 1A, which is 1 user PUT of an 11MB file.

The Average Time charts show the standard deviation.

24.3.1. SCP Reference Tests

The SCP tests reported below are included to provide a point of reference for the repository tests that follow.

The same 11MB file was copied between the client and server using scp. There was no other foreground or background activity on the systems. The first table shows the results of copies from the client to the server (upload); the second from the server to client (download).

For the 1 user test 500 files were copied. For the 5 user test, each user copied 100 files for a total of 500.

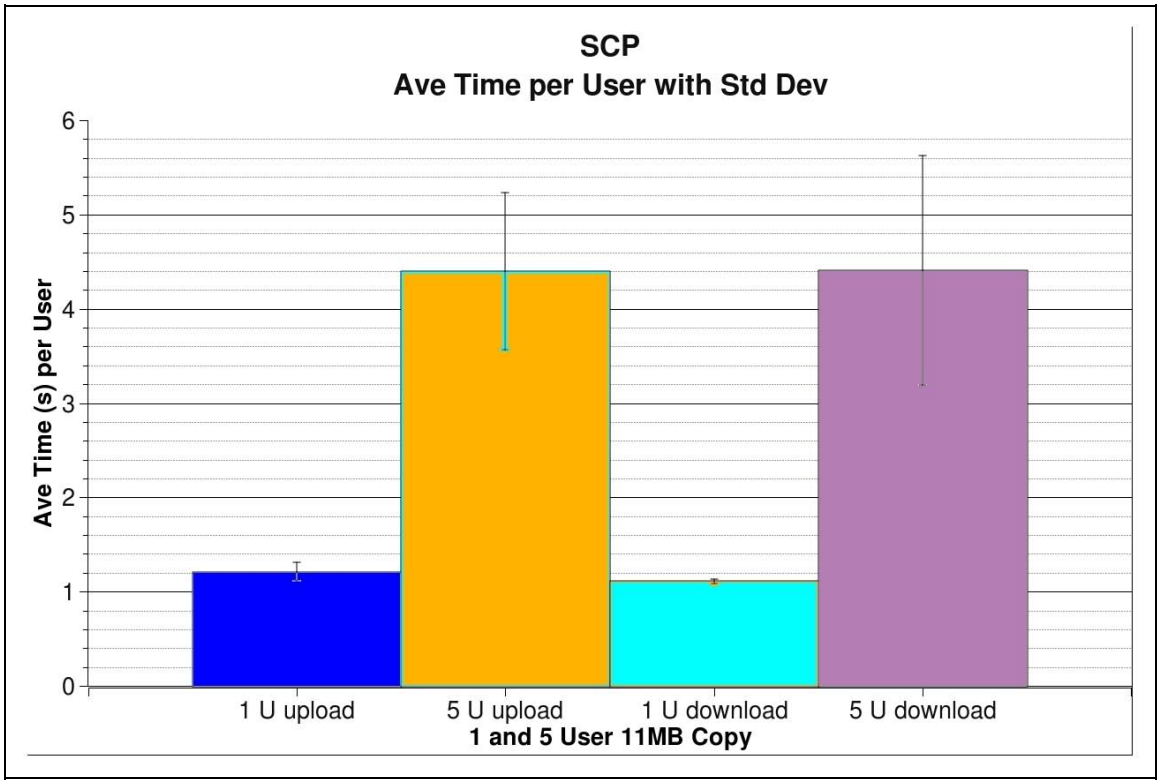
The 50 user scp copy test was not executed due to ssh daemon configuration limitations.

Table 11: SCP Copy from Client to Server (Upload)

Users	File size	Number of Files	Average Time (s)	Std Dev	Throughput per User (KB/s)	Total Throughput (KB/s)
1	11MB	500	1.21	0.19	8701.13	8701.13
5	11MB	500	4.40	1.67	2391.54	11957.70

Table 12: SCP Copy from Server to Client (Download)

Users	File size	Number of Files	Average Time (s)	Std Dev	Throughput per User (KB/s)	Total Throughput (KB/s)
1	11MB	500	1.11	0.04	9485.01	9485.01
5	11MB	500	4.41	2.43	2386.85	11934.25



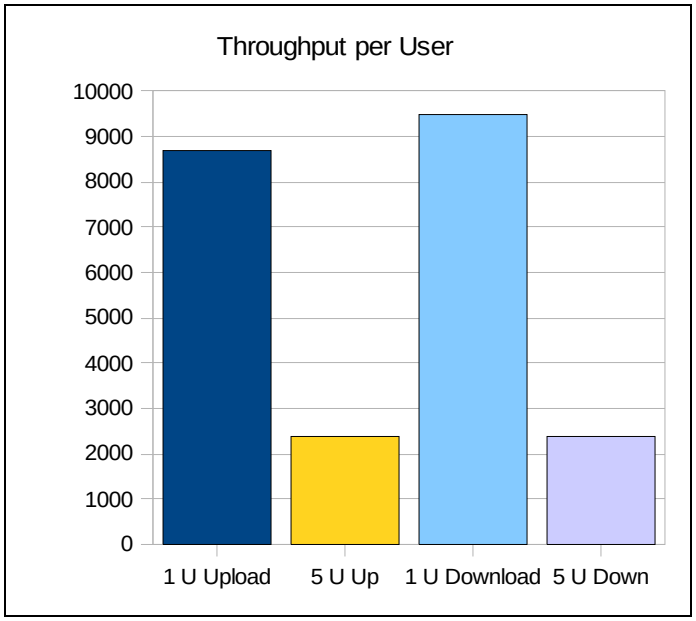


Figure 13: SCP Reference Test 11 MB File Copy

24.3.2. Upload – 1, 5 and 50 Users

An 11MB PDF file and random size data files were uploaded. Multiple test runs were performed; each test ran for 60 minutes. For the 1 and 5 user tests the repository started empty and ended with more than 36K objects. The repository was re-initialized before starting the 50 user tests. The indexer completed (i.e. caught up with the transaction queue) between 1 and 2 hours after the test completed.

The test case tables are interpreted as follows. For the test case 1B-1:

- There were 5 total users.
- A total of 6988 files were uploaded, or 1398 files per user on average.
- The average upload time is 1.73 seconds per user per file.
- The goodput is a measure of the throughput excluding protocol overhead and retransmissions. This was captured for each user and is shown per user.

Table 13: Upload – 1, 5 and 50 Users

Group/ Case	Users	Total Number of Files	Average Time (s) per User	Std Dev	Goodput (KB/s) per User	Notes
SCP	1	500	1.21	0.19	8701.13	For reference.
SCP	5	500	4.40	1.67	11957.70	
1A-1	1	2294	1.02	0.53	2062.97	11MB file.
1A-2	1	2487	0.93	0.43	2238.68	11MB file.
1A-1	1	2123	1.09	0.98	1930.86	Random file sizes.
1A-2	1	2264	1.05	0.78	1993.59	Random file sizes.
1B-1	5	6988	1.73	1.04	1213.91	11MB file.
1B-2	5	7264	1.67	0.98	1255.82	11MB file.
2B-1	5	7723	1.58	0.77	1331.82	Random file sizes.
2B-2	5	6633	1.82	1.17	1128.68	Random file sizes.
1C-1	50	10000	12.31	5.76	151.46	11MB file.
1C-2	50	8773	13.98	7.11	138.76	11MB file.
2C-1	50	8894	13.85	9.04	138.67	Random file sizes.
2C-2	50	8176	15.12	9.21	129.81	Random file sizes.

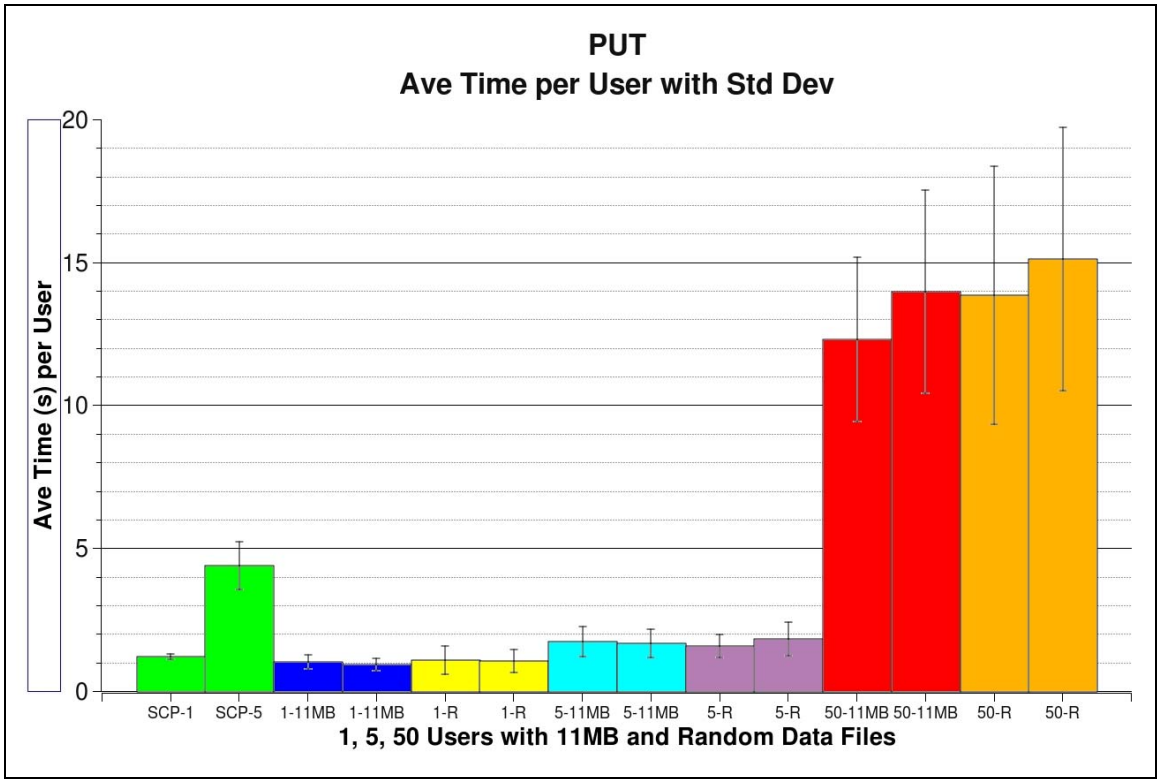


Figure 14: Average Time per User using 11 MB and Random Data Files

In this graph the green bars are the scp reference test results. Otherwise the bars of the same color bars are different runs of the same test. The 11-1 refers to the 11MB file, single user PUT test; 1-R refers to the random data file, single user test.

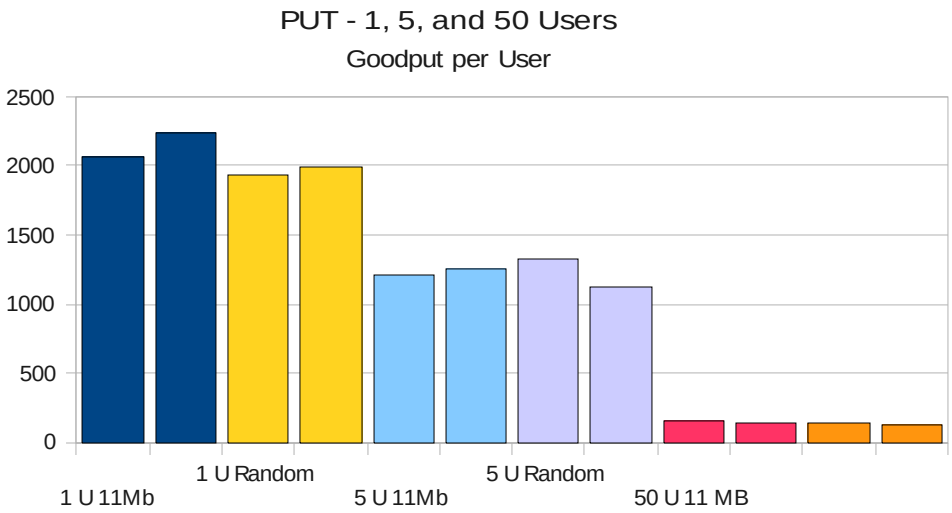


Figure 15: Goodput per User using 11 MB and Random Data Files

24.3.3. Download – 1 and 5 Users

An 11MB PDF file and random size data files were downloaded from the repository which contained approximately 36K objects. Each test ran for 60 minutes.

Table 14: Download – 1 and 5 Users

Group/ Case	Users	Total Number of Files	Average Time (s) per User	Std Dev	Goodput (KB/s) per User	Notes
SCP	1	500	1.11	0.04	9485.01	For reference.
SCP	5	500	4.41	2.43	11934.25	
1D-1	1	2575	1.39	0.19	7574.36	11MB file.
1D-2	1	6019	0.58	0.37	3498.47	11MB file.
2D-1	1	2614	1.37	0.16	7690.55	Random file sizes.
2D-2	1	6011	0.58	0.37	3571.44	Random file sizes.
1E-1	5	3959	4.54	1.18	11592.56	11MB file.
1E-2	5	14865	1.18	0.86	8866.26	11MB file.
2E-1	5	3917	4.57	1.06	11518.99	Random file sizes.
2E-2	5	14808	1.19	0.85	8802.69	Random file sizes.
1F-1	50	4048	39.99	8.76	151.46	11MB file.
1F-2	50	4035	44.73	7.07	138.76	11MB file.
2F-1	50	18834	9.42	9.43	138.67	Random file sizes.
2F-2	50	16603	10.84	7.85	129.81	Random file sizes.

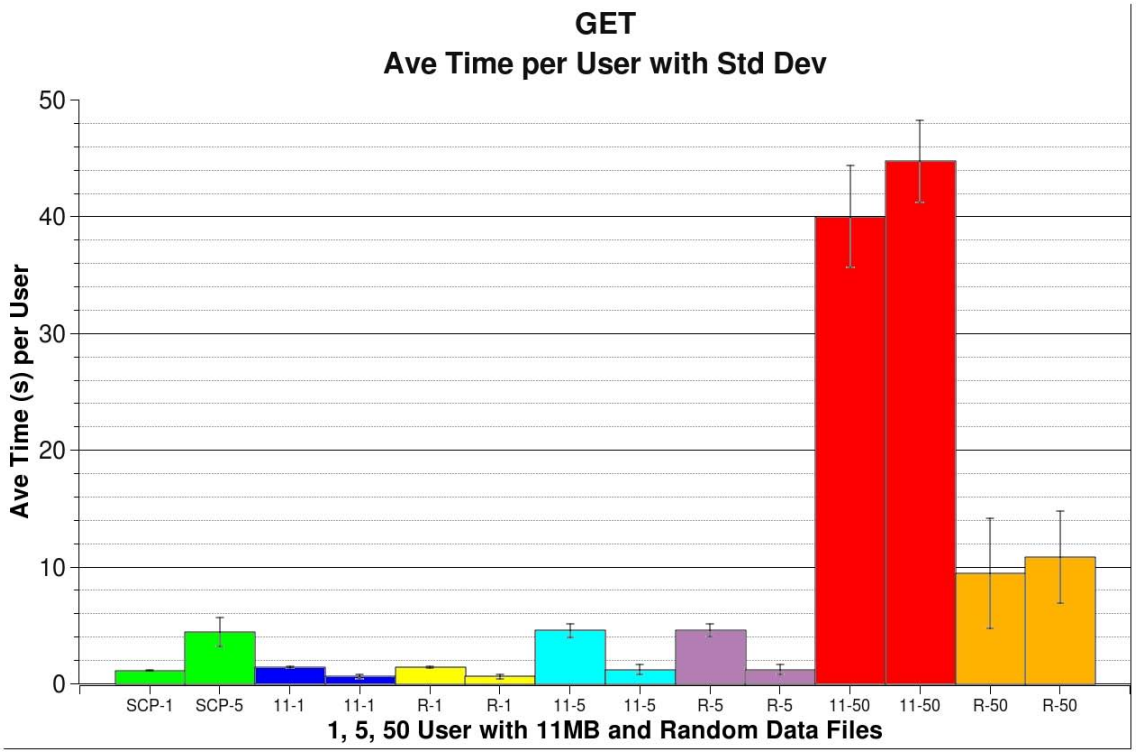


Figure 16: Average Time per User using 11 MB and Random Data Files

In this graph the green bars are the scp reference test results. Otherwise the bars of the same color bars are different runs of the same test.

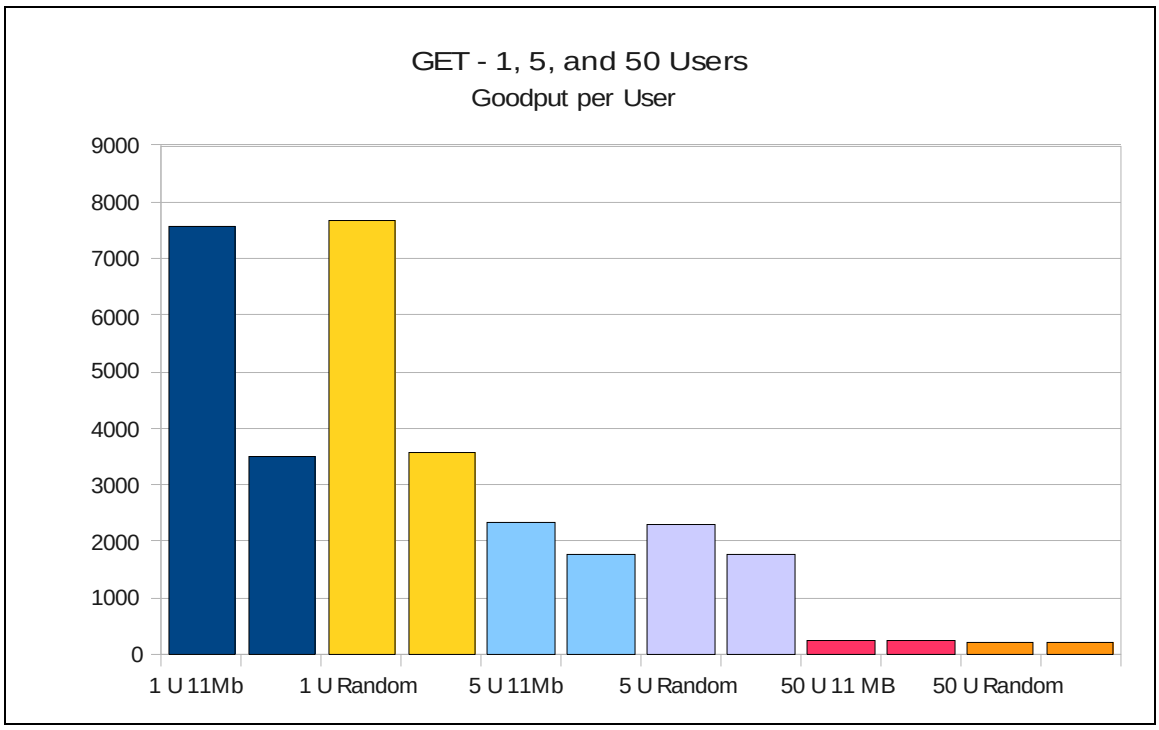


Figure 17: Goodput per User using 11 MB and Random Data Files

24.3.4. Search – 1 and 5 Users

The search operation was performed for a term added to the object attribute during upload. Each test ran for 60 minutes. In general the file size should not be a factor in the search timing; the 2G tests were performed with the random files. For test 2G, the repository contained 8342 objects. Note that the 0 search results were due to a unique query on a data attribute which did not match (the query term was typed incorrectly).

Table 15: Search – 1 and 5 Users

Group/ Case	Users	Total Number of Searches Performed	Number of Search Results	Average Time (s) per User	Std Dev	Notes
1G-1	1	1803	2487	1.63	0.82	11 MB File
2G-1	1	1974	2264	1.52	0.43	Random files
1H1	5	1548	1440	10.97	2.14	11 MB File
2H-1	5	1564	1400	10.82	3.91	Random files
1I-1	50	2161	range	84.52	20.25	11MB file. Range a search results: 9,10,916, 917, 919, 972, 973, 1389.
2I-1	50	3063	range	58.54	24.11	Results 76-86, 0 results for users68, 69, and user63. User63 completed after 1690 s.

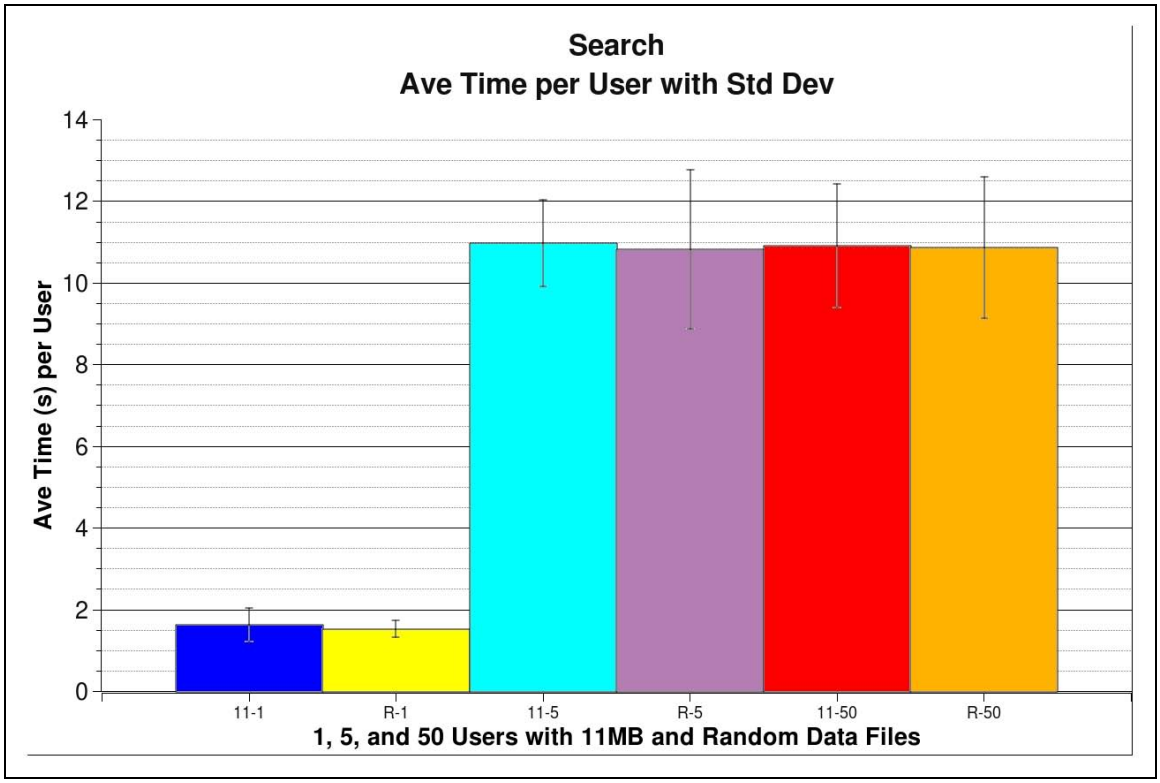


Figure 18: Average Time per User Using 11 MB and Random Files

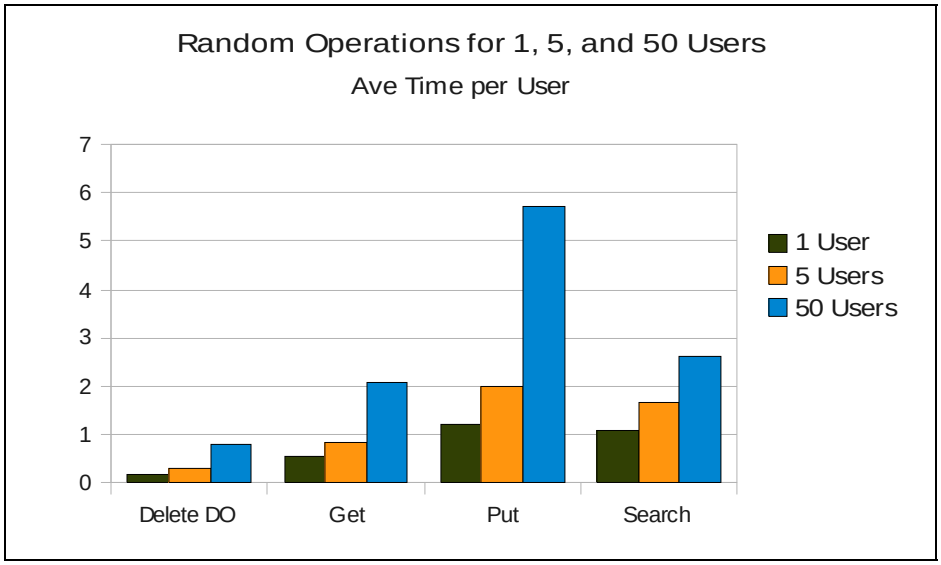
24.4. Test Results – Random Data / Random Operations

24.4.1. 1, 5 and 50 Users

Random files sizes and types were used. The test started with 1438 objects and ended with 39213 objects in the repository. Note that there are no goodput results for the delete and search operations as no (measurable) data is transmitted.

Table 16: Random Operations - 1, 5 and 50 Users

Group/ Case	Us-ers	Operation	Number of Iterations	Average Time (s) per User	Std Dev	Goodput (KB/s) per User
3A	1	Delete	936	0.18	0.27	N/A
		Get	915	0.55	0.37	3466.26
		Put	986	1.23	1.10	1584.35
		Search	982	1.09	3.93	N/A
3B	5	Delete	3038	0.31	0.39	N/A
		Get	2918	0.84	0.91	2415.98
		Put	3091	2.02	1.71	1018.79
		Search	3031	1.65	5.06	N/A
3C	50	Delete	5476	0.81	1.27	N/A
		Get	5521	2.10	2.57	971.57
		Put	5973	5.73	5.38	386.77
		Search	5392	2.63	5.73	N/A



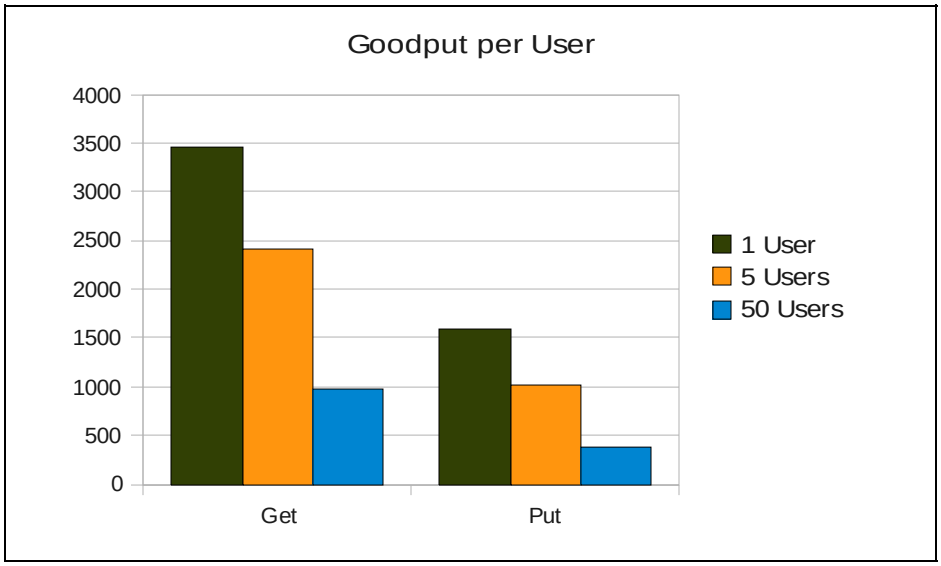


Figure 19: Random Operations using Random Data

24.5. Test Results – Duration Tests

Random files sizes and types were used.

24.5.1. 1 Day Test

Table 17: 1 Day Test - 1, 5 and 50 Users

Operation	Number of Iterations	Average Time (s)	Std Dev	Goodput (KB/s) per User
Delete	35776	0.37	0.57	N/A
Get	35681	0.84	0.90	11945.04
Put	36644	2.13	1.95	4845.9
Search	35362	7.23	18.85	N/A

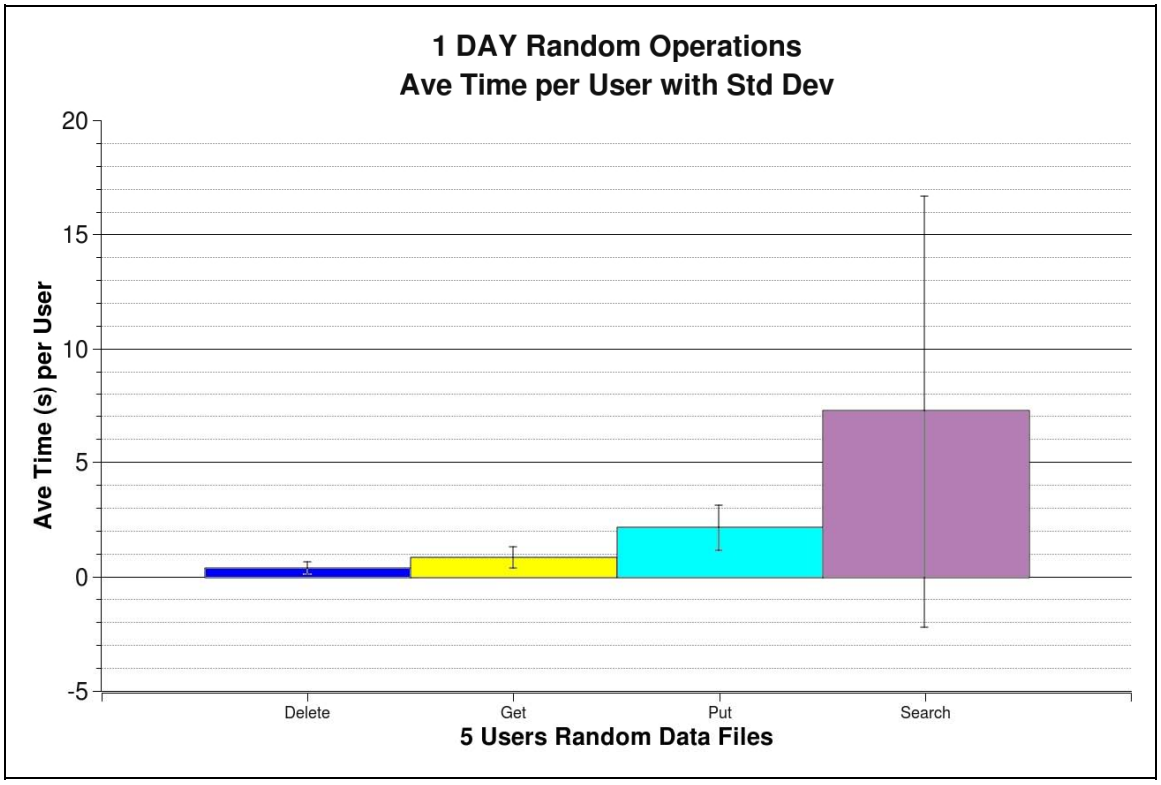


Figure 20: 1 Day Test – Average Time per User for Random Operations

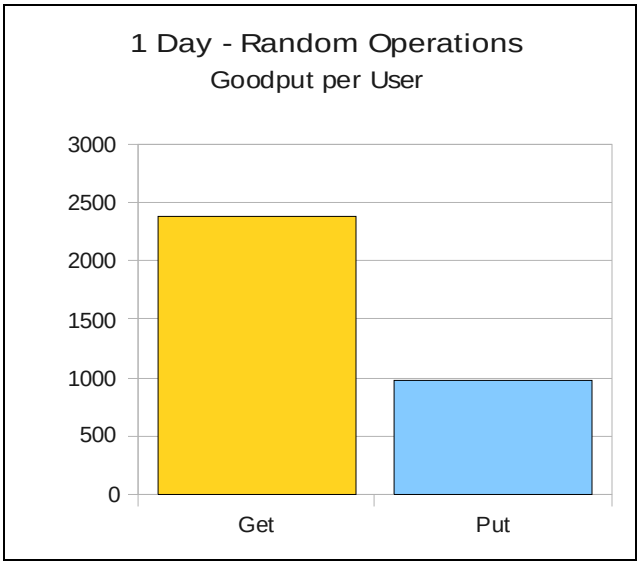


Figure 21: 1 Day Test – Goodput per User for GET and PUT

The following chart shows the average time for each operation per hour. All times are consistent except for Search. Note that the Search data point at Epoch 18 was off the chart at 106.41s. There was insufficient historical data to offer an explanation for this anomaly.

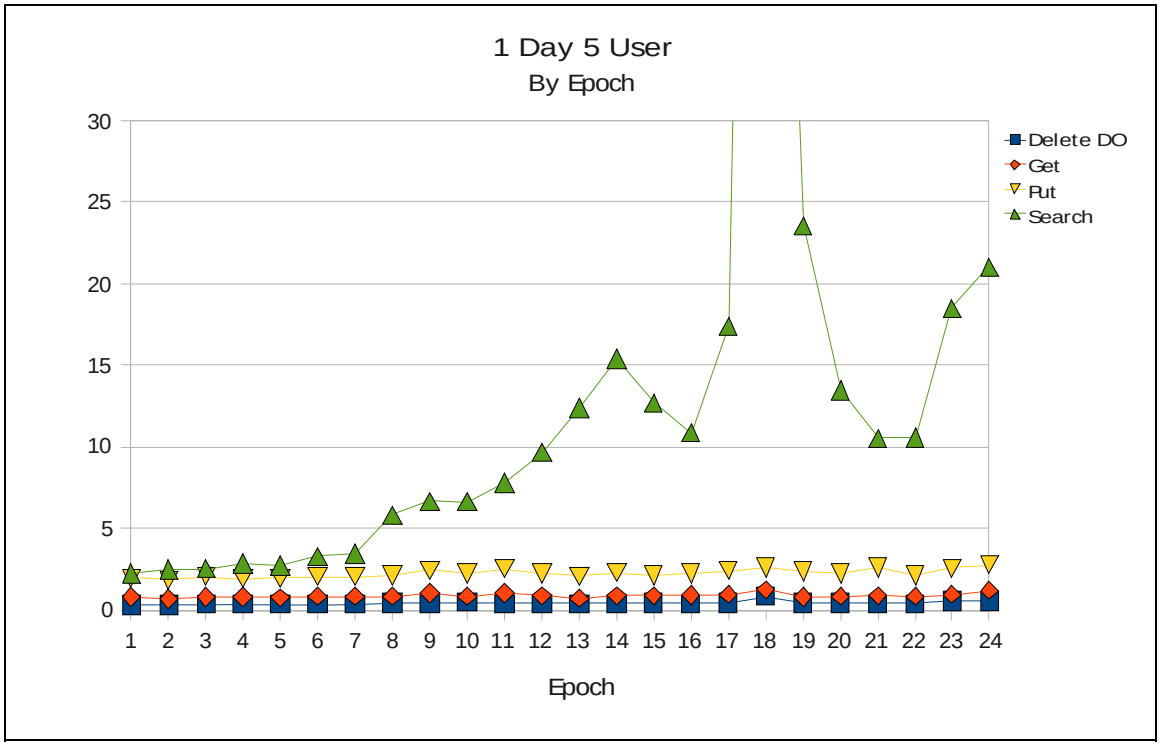


Figure 22: 1 Day Test – Operation per Hour

24.5.2. 5 Day Test

Table 18: 5 Day Test

Operation	Number of Itera-tions	Average Time (s)	Std Dev	Goodput (KB/s) per User
Delete	236327	0.39	0.71	N/A
Get	235240	0.77	0.72	2610.61
Put	237544	2.08	1.80	969.51
Search	233403	4.62	12.62	N/A

24.5.3. 10 Day Test

This test was not executed due to time limitations.

25. CONCLUSIONS

The DO Repository version 1.17-7 was evaluated in terms of performance for several common operations such as put, get, search and delete. These operations were tested based on a previously agreed test plan, and included several conditions as explained before, including testing periods which extended over several days. Even though it's difficult to characterize these operations in terms of real usage patterns, this evaluation proved to be a valuable tool to assess and therefore correct several issues that have not been identified during the development phase.

At this point, comparing the timing observed for the operations between one and five users, the behavior of the DO repository is comparable with accessing a network drive, as shown by the reference to scp. When running 50 users, we observed increased duration time to execute an operation per user. Further investigation is required to determine if the observed 50 user performance is comparable to scp performance, and if the repository can be optimized to increase performance for large numbers of simultaneous users.

With the several day tests, the most significant finding was related to the search operation, where significant degradation in performance was observed for five users after 7 hours. This is under investigation.

Besides the observations mentioned at the beginning of this document, one open question remains from this evaluation, and it is the variability in the measurements as shown by i) the standard deviations over the same tests ii) differences over test runs, and iii) differences when the same operation is repeated or under random operations. This will be investigated.

In conclusion, as measured in this set of tests the DO repository version 1.17-7 behaves adequately for the operations of put, get and delete for five simultaneous users over extended periods of times using different patterns of operations and file sizes. Search operations also behave properly for five simultaneous users, but it was shown that over extended periods of time, it presents an inexplicable behavior.

26. TEST DATA

26.1. Test Files

These files were used for the upload tests.

Table 19: Test Files

File name	Size (bytes)	Notes
1b.txt	1	When used in get and put tests these small files had poor throughput; this was deemed unrealistic so they were removed from the random test executions.
10b.txt	10	
100b.txt	100	
This is a very_long_(FILE-NAME) with +^misc~characters...txt	1023	
1k.gif	1024	
a-p_p!\@e.jpg	1811	
f[r]{u}1t.jpeg	1811	
The Thinker.jpg	4653	
10k.txt	10240	
b#u%g\$.jpg	15347	
InS3c=ts.jpg	15347	
100k.txt	102400	
huckfinn.txt	622888	
10M.tar	10781042	
ngn_jan2009.pdf	11008000	

27. LESSONS LEARNED

27.1. Internal Use of the Repository System for Collaboration

Over the course of this project, the team used a Wiki to share documents, track tickets and communicate. However, in retrospect we realize that we missed an opportunity to utilize the developing repository in a real-world setting.

27.2. Allow Existing User Paradigms to Co-exist with New Functionality

Once information is deposited in a repository the only way to retrieve it, if the client does not know the handle associated with it, is through search. This is a paradigm shift from traditional file systems. Its main advantages are: i) data access is based on free-form queries rather than file names and browsing directories, and ii) multiple repositories and workspaces are concurrently accessible.

This has the potential to improve user access to data; however, there are issues to consider:

- It is essential that the indexer be able to extract all allowable and pertinent information from digital objects. Digital objects which cannot be indexed are not accessible via search, and therefore become invisible to users who do not know the object's handle for direct access.

The performance of the indexer has an impact on how quickly the data can be retrieved. The indexing process is not instantaneous. This can be disconcerting to users accustomed to traditional storage systems; however, dynamically updating user interfaces may mitigate this effect.

While addressing these issues is important, we also believe that systems that introduce new paradigms should provide a mechanism for gracefully transitioning from the old paradigm to the new. In the case of the DOR, that could include a view that mimics a files and folders view.

27.3. Ensure that the Core Repository Functionality Can Support Different Use Models

In addition to the technical effort required to build the DOR-SVN and SVN-DOR interfaces, there was a lot of thought and consideration required regarding usability, including how to make the process work smoothly and sensibly for the two user groups.

For example, we expect DO client tool users to locate individual objects through a search interface. Meanwhile, the typical workflow for an SVN user is to check out an entire repository as a working copy, to locate particular files within the repository by their path, and to periodically synchronize between the local working copy and the remote repository. It took thought and multiple iterations to get the SVN client tools to work with the DO Repository in a way that felt comfortable and useful to users familiar with ordinary SVN repositories.

We foresee that as other clients and existing data management systems are considered for interoperability with repositories that similar conceptual issues will arise. People will want to use repositories in different ways. The goals are to ensure that the core repository functionality supports different use models, then to provide people with tools that utilize that flexibility.

28. FUTURE WORK

28.1. Reduce Administrative Complexity

We found that it would be beneficial to streamline both the server installation process and the administration procedures.

We found that repository installation can be a time-consuming process for administrators who are new to the system and that enhanced administration tools are desirable. Errors included those introduced in handle setup and/or repository setup. This process could be made more robust.

- In addition, each repository provides a high level of granularity for setting permissions and access rules. This provides great flexibility; however, it could also represent a potential vulnerability of the system due to its complexity.
- Currently, for a user to gain access to a shared workspace, the user's handle must be added to the internal.rights and internal.forwarding files for that workspace by an administrator. To allow groups to be used, the group must be created (by the user), the private key added to each group member's PDO, and the group handle added to the forwarding rules (by the administrator).
- Every repository will need an administration procedure that sets up, maintains and updates operational rules. For example, assignments of clients to repositories and the operations that these clients can do in the respective repositories are not automatic. In addition, the current repository implementation has a repository-level set of default rights that are applied to objects that don't contain their own internal.rights element, but it is a static set of rights that do not take into account potential differences in policies for different repositories. The core repository functionality should have the flexibility to support different administrative policies and should provide simple mechanisms for administrators to specify those policies.
- Finally, tools to support administrative activities, as well as tools to manage users (add, remove, backup data, assign and monitor quotas, passphrase management, etc.) and to monitor system performance (disk space, network, CPU, memory, etc.) would be beneficial.

28.2. Further Mechanisms to Support Sharing of Digital Objects

Digital object repositories serve as gatekeepers of the operations that can be performed. However, at the same time, every DO also has the element internal.rights that defines the rules of operations for that object. In internal.rights, every operation on an object can be permitted or rejected based on the individual user's identity or based on a group to which they belong. Operations include setting or getting attributes, putting or retrieving data elements, and many more. The level of granularity is quite large, which provides great flexibility.

In a practical setting, the administrative mechanisms to set up, maintain, update and enforce the rules that support this flexibility could become cumbersome and time consuming.

In practice, user interfaces to the DO system have provided a simplified view of the rules, consolidating the operation rights into 'read' and 'write' permissions and hiding the level of detail that is allowed by the system. If the system supported a standard model of simplified access rights then developing DO clients could be made easier.

Mechanisms to support identity discovery (to locate the handle for another user) are also required. The primary issue is that discovering the handle for another user is best done using an out-of-band method such as the other user supplying their handle via an in-person meeting. Alternate methods such as signed communication from trusted third parties that certify the user's name, handle, and other information (title, birthday, etc.) would be useful here.

28.3. Additional Mechanisms for Group Management

The repository architecture provides a facility for establishing membership in a group, not for group management. Group management is external to the architecture; it is specific to individual organizations and how they choose to manage their groups. Group management applications may also be organization-specific.

Groups are currently defined as components which consist of i) a handle with its respective public key and ii) one certificate distributed to each member of the group containing that member's handle and that is signed by the private key which corresponds to the group handle's public key.

This definition is built on existing components of the repository architecture, and with this simple definition groups can grow their number of members indefinitely.

However, there are a few drawbacks to the current implementation:

- With this architecture, the members of a group cannot be easily listed. The ability to determine group membership is dependent on the group management application.
- Whatever system is developed to manage group members will need to be adapted to regularly generate and distribute certificates.
- Revoking group membership is currently less than instantaneous. If a group member has a connection open with a repository and has already been authenticated as being a member of the group, that authentication will be cached with the connection and will not be re-verified until the connection is re-established and the TTL on the cached group handle (containing the group public key, and soon the revocation list) expires, causing the server to re-verify the group credentials.

Example mechanisms for administering groups may need to be defined and implemented and the revocation list mechanism still needs to be implemented.

28.4. Study Efficiency of Indexing and Search

In the current system implementation, the indexing mechanism is decoupled from the repository application. This provides the advantage of being able to incorporate the best state-of-the art solution available at the time. In the current implementation, the Lucene search engine has been chosen for indexing and search.

Extensive evaluation of the Lucene search engine's configuration and settings within the system has not yet been carried out. In addition, in the current packaging of the repository distribution, the search engine configuration is not exposed to the administrator. For future distributions of the repository it may be desirable to expose these settings to the administrator for configuration and tuning.

Size management of the index is needed to control performance of the searches and of the indexing process. The larger the repository (i.e., the more DOs stored) the larger the indexer may get.

Efficient search mechanisms within the index are required to ensure consistent performance as the repository size and/or number of clients grows.

In the current implementation, every repository has its own index and indexer process. For federated searches, the repository forwards the search to the other repositories with which it communicates, and then the indexer process consolidates the results for the client. Here as well, efficient implementations are crucial for performance and scalability.

28.5. Further Study of Encryption Techniques

Encryption can be used to protect the data portions of DOs in case the object, repository, or physical storage is ever compromised. Access to the DOs, whether encrypted or not, during operation is controlled by the permissions specified in the element internal.rights.

Encryption is involved in two levels of the architecture: communication and data encryption. All DOP connections are encrypted by default to ensure private communication as well as the continuous authentication of the entity on the other side of the connection. Connection encryption is performed either by the built-in DOP encryption or, if the client and server allow it, using an SSL wrapper around an unencrypted DOP connection.

Data-level encryption is also available and provides a method for encrypting data elements so that the repository holding the objects does not need to be trusted with access to the encrypted information. The repository's primary responsibility is making sure the encrypted data is accessible and not lost or destroyed. In the current implementation, all the DO attributes are left in clear for indexing purposes. Encryption uses a key generated by the creator. This key is then deposited in the PDO of the creator encrypted with his/her public key. The system could be modified to encrypt the entire object, not just the data elements, so that adversaries couldn't access object attributes such as timestamps, creators, mime types, annotations, etc. However, this would require scalable encrypted indexing.

If the encrypted element deposited in the DO is shared with other identities, the generated key is distributed to the PDOs of these identities – added to the PDO's key ring - encrypted with their respective public keys.

28.5.1. Observations

- Testing to date has shown that the SSL wrapper is significantly slower than the built-in DOP encryption. Further study is needed to discover how to improve the performance of the SSL-based encryption.
- In this implementation there is no mechanism to retrieve/revoke the key once it has been added to a PDO, therefore, there is no specific mechanism to revoke the encryption privileges. Access to the encrypted object can be discontinued by simply re-encrypting the object using a new key (which is the recommended approach), however this doesn't cover the case in which a user has cached a copy of the object encrypted with the original key.

- Since groups do not possess a PDO structure, they can not be used for sharing encrypted data therefore all the sharing of encrypted elements is done at the individual identity level. In other words there is no adequate procedure to relate groups and encrypted elements. We hope to explore using a Proxy Re-encryption service to allow group members to access objects shared with the group. Such a service might also be used to solve the revocation problem as access to encrypted objects would need to occur in real-time.
- There is no high-level mechanism to add a specific key to other identities after an element has been encrypted and deposited. This can be done easily using the lower-level API, but is not yet supported in the high-level API.
- Without the appropriate administrative mechanisms, the oversight of who has an encryption key can be lost.
- Without appropriate administrative mechanisms, when certificates expire (group membership is no longer valid), there are no high level APIs to update these certificates.

28.6. Additional Testing of Repository Mirroring

To support data recovery even when a repository is compromised, mirroring mechanisms have been implemented, but not yet tested. A few vulnerabilities and performance issues in this regard have been identified and reported in our Trac system. In particular, in the current repository implementation, the mirroring mechanism relies upon a transaction log that stores the timestamp of each event. This requires a synchronized clock for all the mirrors to work properly. Attention must be paid to the clocks' synchronization and to the integrity of the files that maintain the last state of the time stamps.

Furthermore, in the current implementation, no mechanism has been created to help a client determine which server in a mirrored service is the most preferable. Such a mechanism could support load balancing; incorporating the current handle system load balancing mechanism would seem appropriate.

28.7. Conduct Usability Studies

New paradigms in user experience are available with the DOA in terms of data sharing, searching as the mechanism of data retrieval, and additional information provided by metadata. After the first users interacted with the preliminary application, it became clear that usability studies are required before a re-architecture of the information display and accessibility is developed and implemented.

REFERENCES

"Handle System Technical Manual, version 2".
Corporation for National Research Initiatives . 30 June 2009
<http://www.handle.net/tech_manual/Handle_Technical_Manual.pdf >.

“Handle System Administration Tools”
Corporation for National Research Initiatives . 30 June 2009
<http://www.handle.net/handle_tool.html>

LIST OF ACRONYMS

DA	Data attribute is a key-value pair of text strings. A data attribute is associated with either a digital object or a data element of a DO.
DE	Data element is a key/value pair contained in a digital object, where the key is used to uniquely address the element within a digital object and the value is a series of bytes.
DO	Digital object is an abstract digital entity consisting of a set of key/value pairs, one of which is its unique identifier. A digital object repository enables operations to be applied to a given set of digital objects, e.g., all or some of the values may be retrieved or manipulated.
DOR	Digital Object Repository acts as a container for digital objects. A DOR authenticates clients and determines which clients are permitted to perform which operations on the digital objects it contains. A DOR also acts as a host for "operators" that perform operations on objects.
Keychain	A keychain is a list of encrypted symmetric keys, stored across one or more data elements within a personal digital object (PDO). The keys have been encrypted using the public key associated with the PDO and can only be decrypted by the holder of the associated private key. In most cases, any user is allowed to append keys to other user's key chains. When the owner of the PDO encounters an encrypted object or data element, they can check their keychain for a key that is able to decrypt the object or data element.
Object ID	A digital object identifier (DO identifier) is a unique Unicode text string that can be resolved to locate a repository through which the client can interact with the DO being identified.
PDO	Personal digital object is a digital object that represents an individual. It can contain a keychain, a set of group membership certificates, and a mapping of object IDs to user-friendly names that is configured by the owner of the PDO.